TAMPEREEN AMMATTIKORKEAKOULU
Sähkötekniikan koulutusohjelma
Automaatiotekniikka

Tutkintotyö

Teija Setola

SIGNALOINNIN TOIMINTOTESTAUKSEN AUTOMATISOINTI

Työn valvoja Lehtori Harri Joki
Työn teettäjä TietoEnator Telecom & Media Oyj, ohjaajana DI Mikko Virtanen
Tampere 2006

TAMPEREEN AMMATTIKORKEAKOULU
Sähkötekniikka
Automaatiotekniikka
Setola, Teija Signaloinnin toimintotestauksen automatisointi
Tutkintotyö 48 sivua + 4 liitesivua
Työn valvoja Lehtori Harri Joki
Työn teettäjä TietoEnator Telecom & Media Oyj, ohjaajana DI Mikko Virtanen
Kesäkuu 2006
Hakusanat signalointi, GSM, regressiotestaus

TIIVISTELMÄ

Signaloinnilla eli merkinannolla suoritetaan puhelinverkon tiedonsiirtoon liittyviä tehtäviä, kuten puhelun muodostus, ylläpito ja purku. GSM (Global Systems for Mobile Telecommunications) on matkapuhelinstandardi joka mahdollistaa puheen ja datan siirron radioteitse matkapuhelinverkossa.

Ohjelmistotestauksen tärkeys on korostunut GSM-verkon signalointien tuotekehityksessä. Yksi ohjelmistotestauksen tasoista on toimintotestaus, jossa keskitytään ainoastaan ohjelman toiminnallisuuksien testaamiseen. Regressiotestaus on uudelleentestausta, jota tarkastellaan tässä työssä toimintotestaustasolla. Regressiotestauksella varmistetaan, että ohjelmaan tullut muutos ei ole vaikuttanut ohjelman muihin osiin. Ohjelmistotestauksen jatkuva tehostaminen on lisännyt tarvetta automatisoinnille. Erityisesti usein toistuvan regressiotestauksen automatisoinnilla säästetään paljon aikaa.

Tämän tutkintotyön tarkoituksena oli signaloinnin toimintotestauksen automatisointi. Ensin kartoitettiin toimintotestauksen nykytilanne automatisoinnin näkökulmasta. Kartoituksen perusteella laadittiin suunnitelma ja kehys toimintotestauksen automatisoimiseksi. Lisäksi tarkoituksena oli laaditun suunnitelman mukaisesti automatisoida valittujen regressiotestitulosten analysointi. Automatisointi toteutettiin testauksen analysointiin suunnitellulla työkalulla. Lopputuloksena saatiin aikaan yhtenäinen toimintamalli signaloinnin testaustulosten analysoinnille. Lisäksi jatkosuunnitelma vielä laajemmalle automatisointiprojektille selkeytyi ja testitulosten analysoinnin automatisointi suoritettiin onnistuneesti valituille regressiotestitapauksille.

TAMPERE POLYTECHNIC
Electrical Engineering
Automation Engineering
Setola, Teija
Automatization of Functional Testing in Signalling
Engineering Thesis 48 pages, 4 appendices
Thesis Supervisor Lecturer Harri Joki
Commissioning Company TietoEnator Telecom & Media Oyj,
supervisor Mikko Virtanen (MSc)
June 2006
Keywords signalling, GSM, regression testing

ABSTRACT

The functions of signalling in telecommunication networks are setting up, supervising and clearing the call. GSM is a standard for mobile telecommunications which provides speech and data transmission by radio communications.

The importance of software testing has been emphasized in GSM network development. Functional testing is a layer of software testing which concentrates on functionalities. Regression testing is repetitive and it is covered in this thesis on functional testing level. Regression testing ensures that modifications to a software system have not harmed the other parts of the software. Software testing should be all the time more effective and that is why the need for automatization has increased. Especially a lot of time can be saved with the automatization of recurrent regression testing.

The purpose of this thesis was the automatization of functional testing in signalling. First the present situation of functional testing was clarified from the viewpoint of automatization. Based on clarification, a plan and a framework were made to automatize functional testing. In addition, the purpose was to automatize the analyzation of the results from the selected regression test cases. Automatization was executed with a tool which has been designed for this purpose. As a result, harmonious operations model was reached for analyzing the test results of signalling. Also the follow-on to the automatization plan was clarified and analyzation automatization for the selected regression test cases was accomplished successfully.

ALKUSANAT

Tämä tutkintotyö on tehty osana työskentelyäni TietoEnator Telecom & Media -osakeyhtiössä. Työn aihe muodostui yrityksen tarpeesta kehittää toimintotestausmenetelmää. Yrityksen puolesta työn ohjaajana toimi esimieheni Mikko Virtanen. Häntä haluaisin kiittää saamistani ohjeista ja asiantuntevasta suhtautumisesta työhön. Koko signalointijaosta haluan kiittää avoimesta vastaanotosta työryhmään. Tampereen ammattikorkeakoulun lehtori Harri Joki toimi koulun puolesta työni valvojana ja häntä haluaisin kiittää joustavuudesta ja kärsivällisyydestä työni aikataulujen suhteen.

Lisäksi kiitoksen ansaitsevat perheeni ja ystävät, joiden tuki on auttanut suunnattomasti tämän työn loppuun saattamisessa.

Tampereella 2. kesäkuuta 2006

Teija Setola

Tammelankatu 1A17
33100 TAMPERE
email: teija.setola@tietoenator.com
puh. +358 40 576 5323

SISÄLLYSLUETTELO

TIIVISTELMÄ **2**

ABSTRACT **3**

ALKUSANAT **4**

SISÄLLYSLUETTELO **5**

LYHENTEET **7**

1 JOHDANTO **9**

1.1 Työn tausta 9

1.2 Työn tavoite ja rajaukset 10

1.3 Työn rakenne 10

2 SIGNALOINTI GSM-VERKOSSA **11**

2.1 GSM-verkko 11

2.2 SS7-signalointijärjestelmä 14

2.3 ISUP-signalointi 17

2.3.1 Signalointistandardit ja niiden historia 17

2.3.2 Sanomasekvenssi 19

3 OHJELMISTON TESTAUS **21**

3.1 Ohjelmiston kehitysprosessi 22

3.2 Ohjelmistovirheet ja niiden kustannukset 25

3.3 Testauksen vaiheet ohjelmiston kehitysprosessissa 27

3.4 Ohjelmistotestaajan tehtävä 29

3.5 Testauksen dokumentointi 31

4 TESTAUKSEN JA TULOSTEN ANALYSOINNIN AUTOMATISOINTI **32**

4.1 Automatisoinnin tarkoitus 32

4.2 Automatisoinnissa huomioitavaa 33

4.3 Automatisoinnin ongelmat 33

5 SIGNALOINNIN TOIMINTOTESTAUKSEN AUTOMATISOINNIN SUUNNITELMA JA TOTEUTUS 34

5.1 Testausympäristö 34

5.2 Signaloinnin toimintotestauksen kuvaus 37

5.3 Toimintotestauksen nykytilanne 39

5.4 Automatisoinnin kehitysprojekti 40

5.5 Automatisoinnin kehys 41

5.6 Regressiotestitapausten automatisointi 43

6 PÄÄTELMÄT 46

LÄHTEET 47

LIITTEET 49

Liite 1. IDA2-AA sanomamonitorointi ja automatisointiskripti 49

Liite 2. IDA2-AA testitoimenpide 50

Liite 3. IDA2-AA ohjaustiedoston testiskripti 51

Liite 4. IDA2-AA raportti 52

LYHENTEET

ACM	Address Complete Message, numerot vastaanotettu -sanoma
ANM	Answer Message, vastaussanoma
BICC	Bearer Independent Call Control
BSC	Base Station Controller, tukiasemaohjain
BSS	Base Station Subsystem, tukiasemajärjestelmä
BSSAP	Base Station Subsystem Application Part
BTS	Base Transceiver Station, tukiasema
CAS	Call-associated Signalling, kanavakohtainen merkinanto
CCSU	Common Channel Signalling Unit, signalointiyksikkö
GSM	Global System for Mobile communications
HIT	Holistic Integration Tester
HLR	Home Location Register, kotirekisteri
IAM	Initial Address Message, aloitusosoitesanoma
IDA2-AA	Integrated DX200 Analyzer - Automated Analysis
ISDN	Integrated Services Digital Network, digitaalinen monipalveluverkko
ISO	International Standardization Organization, kansainvälinen standardointijärjestö
ISUP	ISDN User Part, YKM:n protokolla
ITU-T	International Telecommunication Union – Telecommunications Standards Sector, kansainvälinen televiestintäliitto
IUP	Interconnect User Part, merkinantoprotokolla
LAPD	Link Access Procedure on the D-channel
MAP	Mobile Application Part, matkapuhelinosa
MML	Man-Machine Language, standardoitu ohjauskieli
MS	Mobile Station, matkapuhelinasema
MSC	Mobile services Switching Center, matkapuhelinkeskus
MTP	Message Transfer Part, sanomansiirto-osa
MTP3	Message Transfer Part level 3
NMS	Network Management Subsystem, verkonhallintajärjestelmä
NSS	Network Switching Subsystem, verkon alijärjestelmä
NUP	National User Part, kansallinen puhelinkäyttäjäosa

O&M	Operations & Maintenance, käytönohjaus
OSI	Open Systems Interconnection, avointen järjestelmien yhteenliittämismalli
PBX	Private Automatic Branch Exchange, automaattinen tilaajavaihde
PC	Personal Computer, henkilökohtainen tietokone
PCM	Pulse Code Modulation, pulssikoodimodulaatio
PSTN	Public Switched Telephone Network, yleinen puhelinverkko
REL	Release Message, purkusanoma
RLC	Release Complete, purkamisen kuittaus
RS232	Standardoitu liitäntätapa
SCCP	Signalling Connection and Control Part, merkinantoyhteyden ohjausosa
SS7	Common Channel Signalling System No.7, merkinantojärjestelmä nro 7
STP	Signalling Transfer Point, merkinannon siirtopiste
TC	Transcoder, muunnin
TCAP	Transaction Capabilities Application Part, tapahtuman käsittelyosa
TCP/IP	Transmission Control Protocol/Internet Protocol, tiedonsiirtokäytäntö
TTCN	Testing and Test Control Notation
TUP	Telephone User Part, puhelinpalvelua varten määritelty käyttäjäosa
VLR	Visitor Location Register, vierasrekisteri
YKM	Yhteiskanavamerkinanto

1 JOHDANTO

1.1 Työn tausta

Signalointia käytetään GSM-verkossa yhteyden hallinnointiin. Puhelinliikenteessä on olemassa eri signalointeja, joista tässä työssä keskitytään tarkemmin ISUP-signalointiin (ISDN User Part). ISUP on signalointiprotokolla, jota matkapuhelinkeskukset käyttävät saadakseen puhelun yhdistettyä piirikytkentäisen tietoliikenneverkon läpi.

Ohjelmistotestauksen tärkeys on korostunut signaloinnin tuotekehityksessä. Tarkoituksena on löytää ohjelmistoviat mahdollisimman aikaisessa tuotekehityksen vaiheessa. Näiden vaiheiden mukaan jakautuneesta testauksesta käsitellään tässä työssä ainoastaan toimintotestausta. Toimintotestauksen tehtävänä on testata toiminnallisia ominaisuuksia, jotka koostuvat yhdestä tai useammasta ohjelmalohkosta. Toimintotestauksessa testauskohteen toteutusta ei tunneta, vaan sitä käytetään pelkästään määritellyn rajapinnan toimintojen kautta. Tuotekehityksen aikana keskitytään pääasiassa uusien ominaisuuksien ja toiminnallisuuden toteuttamiseen ja testaukseen. Ohjelmistossa on kuitenkin huomattava määrä vanhoja ominaisuuksia, joita ei muuteta uusissa julkaisuissa. Näiden vanhojen ominaisuuksien oikea toiminta varmistetaan regressiotestauksella. Vaikka testaus voidaan suorittaa osittain manuaalisesti ensimmäisellä kertaa, se ei kuitenkaan tue toistettavaa ja johdonmukaista regressiotestausta. Tästä syystä regressiotestaus on hyvä automatisoida.

1.2 Työn tavoite ja rajaukset

Tämän tutkintotyön tavoitteena on signaloinnin toimintotestauksen automatisointi. Tarkoituksena on ensin kartoittaa signaloinnin toimintotestauksen nykytilanne automatisoinnin näkökulmasta. Automatisoinnilla tarkoitetaan tässä yhteydessä ohjelmistotestauksen sekä testitulosten analysoinnin suorittamista automaattisesti ohjelman avulla. Kartoituksen perusteella laaditaan suunnitelma ja kehys toimintotestauksen automatisoimiseksi. Kehyksen mukaisesti automatisoidaan valittujen regressiotestitapausten tulosten analysointi. Työ rajataan käsittelemään ainoastaan signaloinnin toimintotestausta ja toimintotestauksen osana olevaa regressiotestausta. Automatisointi rajataan vain testitulosten analysointiin. Näiden rajausten ulkopuolelta käsitellään lyhyesti toimintotestauksen automatisointia myös testitapausten suorituksen osalta. Tulosten raportoinnin automatisointia ei käsitellä.

1.3 Työn rakenne

Tutkintotyön sisältö jakautuu seuraavasti. Luvussa 2 tarkastellaan tarkemmin GSM-verkon rakennetta sekä signalointia GSM-verkossa. Samassa yhteydessä esitellään signalointiverkon eri komponentit ja protokollat sekä protokollien suhde kansainväliseen ”OSI-malli” -standardiin. Lisäksi tarkastellaan ISUP-signaloinnin sanomarakennetta. Luku 3 käsittelee ohjelmistotestausta yhtenä tuotekehityksen osana. Samassa luvussa kerrotaan hieman tarkemmin toimintotestauksesta ja regressiotestauksesta. Luvussa 4 käydään läpi testauksen automatisointiin liittyviä asioita yleisesti. Luvuissa 2, 3 ja 4 luodaan lähdekirjallisuuden avulla viitekehys, jonka pohjalta luvun 5 signalointitestauksen nykytilanteen analysointi ja automatisointisuunnitelman laadinta suoritetaan. Viimeisessä luvussa kootaan yhteen saadut tulokset, tarkastellaan automatisoinnin tuomia hyötyjä ja käydään läpi tulevaisuuden kehitysmahdollisuudet.

2 SIGNALOINTI GSM-VERKOSSA

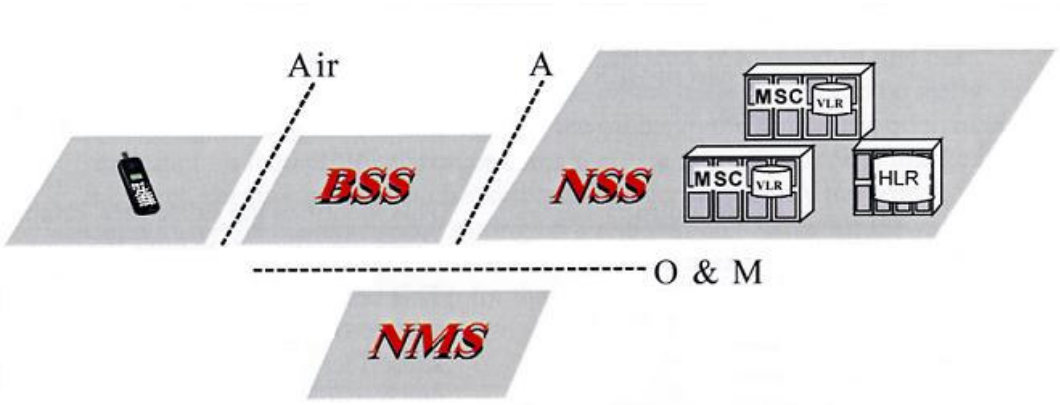
Tässä luvussa käsitellään GSM-verkon rakenne yleisellä tasolla sekä signaloinnin toteuttaminen ja toiminta verkon kannalta. Lisäksi selvitetään lyhyesti signalointiin tarvittavat verkkoelementit, protokollat ja niiden funktiot. Lopuksi kerrotaan tarkemmin ISUP-signaloinnin historiasta, standardeista ja sanomasekvenssistä. Signalointia tietoliikenteessä voidaan verrata yksinkertaisimmillaan vanhanaikaiseen vaihteeseen, jossa henkilö yhdistää puheluita konkreettisesti johtoja yhdistelemällä. Puhelun aikana vaihteen henkilö tarkkailee ja ylläpitää puhelua. Puhelut puretaan taas irrottamalla johdot. Nykyään vaihteen tehtävät hoitavat automaattiset digitaaliset keskukset.

2.1 GSM-verkko

Matkapuhelinverkko rakentuu periaatteessa puhelimista, tilaajaverkosta, puhelinkeskuksista ja erilaisista keskustenvälisistä yhteyksistä /3/. GSM-verkko on piirikytkentäinen, jolloin liikennöivien osapuolien välille muodostetaan pysyvä yhteys tiedonsiirron ajaksi. Yhteyttä varten pitää ensin varata siihen tarvittavat resurssit. Tiedonsiirron jälkeen yhteys päätetään ja resurssit vapautetaan verkon käyttöön. Piirikytkentäistä verkkoa voidaan käyttää sekä puheen- että datan-siirtoon.

GSM-verkko voidaan jakaa keskusjärjestelmään (NSS, Network and Switching Subsystem), tukiasemajärjestelmään (BSS, Base Station Subsystem) sekä näitä hallitsevaan ja ohjaavaan käytön hallintojärjestelmään (NMS, Network Management Subsystem) /5/. Kuvassa 1 on havainnollistettu näitä alijärjestelmiä ja niiden välisiä rajapintoja. Itse verkko puheluiden muodostamista varten muodostuu NSS:stä ja BSS:stä. BSS on vastuussa radiotien kontrolloimisesta, ja kaikki puhelut yhdistyvät BSS:n kautta. NSS huolehtii puheluiden hallinnasta ja kytkemisestä.

Puhelut yhdistyvät aina NSS:n toimesta ja sen kautta. NMS on verkon operaatio ja ylläpito-osa, jota tarvitaan kontrolloimaan koko GSM-verkkoa. Verkko-operaattori tarkkailee ja ylläpitää verkon laatua ja palvelua NMS:n kautta /1/. Puhelimien ja BSS:n välissä on ilmarajapinta, BSS:n ja NSS:n välissä taas A-rajapinta ja NMS:ään on O&M-rajapinta (Operations & Maintenance).

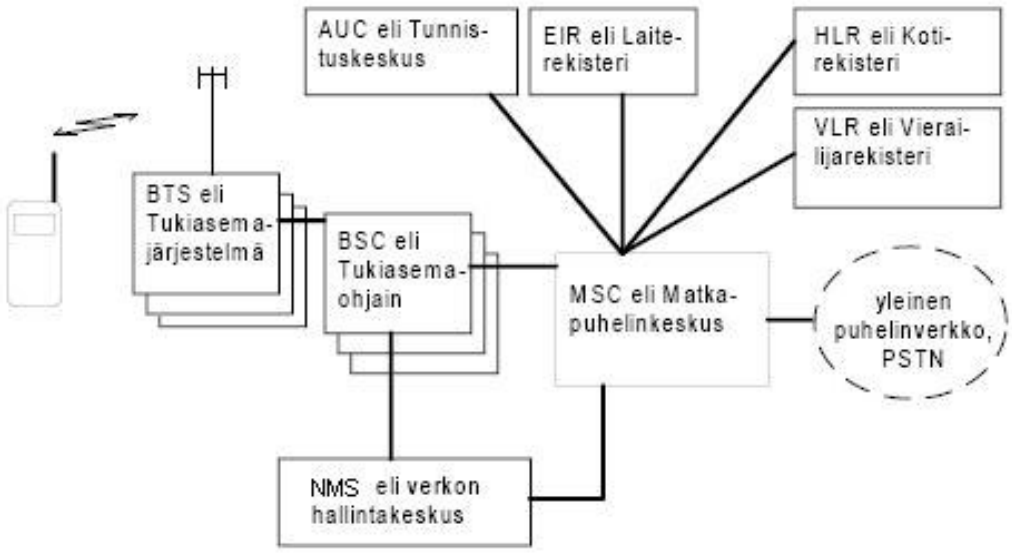


Kuva 2.1 GSM-verkon alijärjestelmät /1/

Tukiasemajärjestelmä BSS koostuu kolmesta verkkoelementistä, tukiasemista (BTS, Base Tranceiver Station), näitä hallitsevista tukiasemaohjaimista (BSC, Base Station Controller) ja muuntimista (TC, Transcoder). /5/ Tukiasema sisältää radiolähettimet ja mahdollistaa yhteyden matkapuhelimiin sekä ylläpitää ilmarajapintaa. Tukiasemaohjain on keskeinen tukiasemajärjestelmän verkkoelementti, joka kontrolloi radioverkkoa. Ohjaimen vastuulla on mm. yhteyden muodostaminen matkapuhelimen ja keskusjärjestelmän NSS välille. Muunnin huolehtii taas puheen konvertoimisesta. /1/

Kuvasta 2.1 nähdään myös, että NSS sisältää matkapuhelinkeskuksia (MSC, Mobile services Switching Centre) ja kotirekistereitä (HLR, Home Location Register). Matkapuhelinkeskuksiin on integroitu vierasrekisterit (VLR, Visitor Location Register). Matkapuhelinkeskuksen tärkeimpänä tehtävä on kytkeä GSM-verkkoon tulevat puhelut tukiasemaohjaimen ja tukiaseman kautta oikealle tukiasemajärjestelmälle sekä BSS:ltä tulevat puhelut muihin verkkoihin /5/. Tässä

työssä puhuttaessa keskuksesta tai MSC:stä, tarkoitetaan matkapuhelinkeskusta. Puhelinkeskuksia on useita erilaisia. Niiden keskeisimmät toiminnot ovat tilaajien liittäminen puhelinverkkoon, keskusten yhdistäminen toisiin keskuksiin ja luonnollisesti puheluiden yhdistäminen. Puhelinverkkoon rakentuu hierarkia, jossa on useita eri tasoilla toimivia puhelinkeskuksia. Kun soitetaan esimerkiksi GSM-puhelimella tavalliseen lankapuhelinverkkoon, on nimenomaan keskuksen tehtävänä huolehtia tarvittavista merkinanto- ja protokollamuutoksista. /3/



Kuva 2.2 GSM-verkon arkkitehtuuri /16, muokattu/

Havainnollistetaan GSM-verkon rakennetta esimerkin kautta. Koska matkapuhelin voi sijaita missä tahansa, verkon pitää saada tietää, missä matkapuhelin sijaitsee. Tämä onnistuu radiolinkin avulla. Verkko ylläpitää sijaintitietoja monen tietokannan avulla. /1/

Kun käyttäjä kytkee matkapuhelimensa päälle alueella, jossa on paikallisen operaattorin ylläpitämä verkkopalvelu, alue yhdistyy ilmarajapinnan kautta vierasrekisteriin (VLR). Vierasrekisteri on integroituna matkapuhelinkeskukseen (MSC) ja sisältää matkapuhelinkeskuksessa vierailevien matkapuhelinten

tilaajatiedot. Kun tilaaja liikkuu toiseen VLR-alueeseen, hänen tietonsa pyyhkiytyvät vanhasta VLR:stä ja tallentuvat uuteen, sijaintia vastaavaan VLR:ään Samalla HLR:ää informoidaan uudesta VLR-osoitteesta. Käyttäjän oman operaattorin on myös tiedettävä tilaajan sijainti. Tätä varten on oma kotirekisteri (HLR). HLR sisältää tilaajatiedot pysyvällä tasolla. Ainoa muuttuva tieto HLR:ssä on tilaajan VLR-osoite. /1/

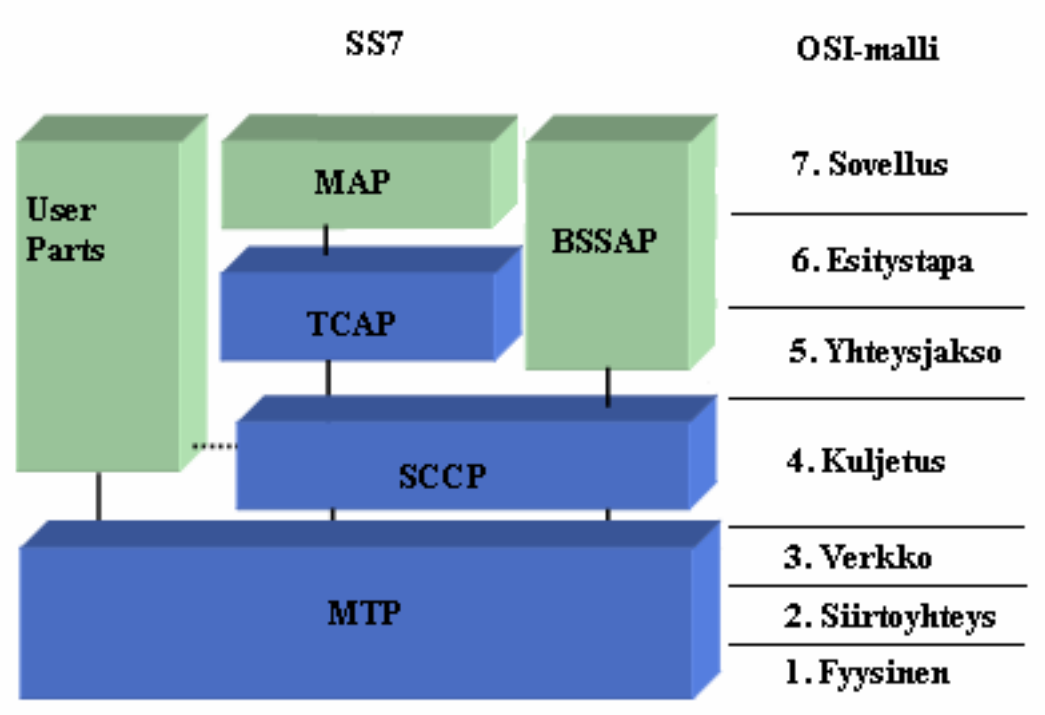
Koska GSM-verkko on solukkopohjainen järjestelmä, niin kerran muodostettu puhelu ei katkea, vaikka puhelin liikkuu solukkoverkon alueella, vaan yhteys siirtyy solusta toiseen. Jokaisen solun sisällä on tukiasema. Kun sijainti muuttuu, tilaajan HLR:ää informoidaan uudesta VLR-osoitteesta. /1/

2.2 SS7-signalointijärjestelmä

Keskusten välinen merkinanto pohjautuu kansainvälisen televiestintäliiton ITU-T:n (International Telecommunication Union – Telecommunications Standards Sector) määrittelemään Common Channel Signalling System nro 7:n eli SS7-merkinantojärjestelmään. SS7 on sanomapohjainen merkinantojärjestelmä /17/. Eri signalointeja ovat mm. ISUP (ISDN User Part), TUP (Telephone User Part), IUP (Interconnect User part), PBX (Private Automatic Branch Exchange), BICC (Bearer Independent Call Control) ja CAS (Call-associated Signalling). Tässä työssä kerrotaan tarkemmin ainoastaan ISUP-signaloinnista, koska signaloinnin testaus keskittyy ISUP:iin.

Signalointijärjestelmässä olevien laitteiden on noudatettava yhteyskäytäntöä eli protokollaa, jotta niiden välinen yhteys olisi mahdollinen. Tietoliikennetekniikassa käytetään tavallisesti yhtä aikaa useita eri protokollia, jotka huolehtivat kukin omasta tarkoin rajatusta tehtävästään siirrossa. Yhdessä nämä protokollat muodostavat protokollapinon. Kansainvälinen standardointijärjestö ISO (International Standardization Organization) on standardoinut tietoliikennetekniikassa käytettävän protokollapinon eri tehtävät. ISO:n

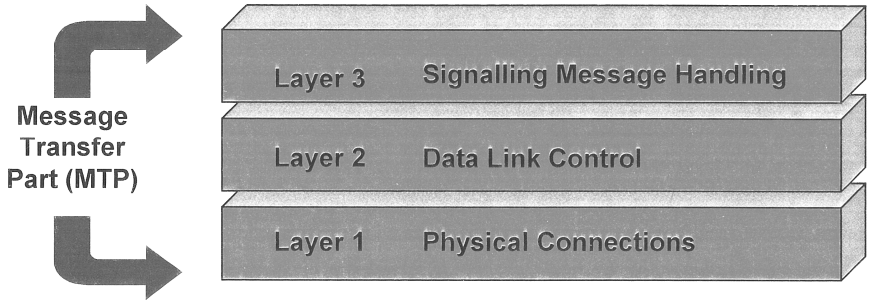
protokollapinoa kutsutaan OSI-malliksi (Open Systems Interconnection) /6/. Kuvassa 2.3 OSI-mallia verrataan GSM-verkkoelementtien kommunikoinnin perustana olevaan SS7-protokollapinoon. GSM-verkon eri elementtien protokollapinot eroavat toisistaan. Kuvassa 3 on esitetty MSC:n protokollakerrokset.



Kuva 2.3 Protokollakerrokset /10, muokattu/

Aluksi käydään läpi lyhyesti OSI-mallin 7 protokollakerrosta. Fyysisen kerroksen tehtävänä ovat toiminnalliset tapahtumat, joiden seurauksena dataa siirretään. Siirtoyhteyskerros tarjoaa keinot luotettavaan tiedonsiirtoon. Verkkokerroksen avulla tiedonsiirto reitittyy jotain kommunikointiverkkoa käyttämällä. Kuljetuskerros tarjoaa mekanismit tiedon välittämiseksi kahden järjestelmän välillä. Yhteysjaksokerros mahdollistaa kuljetuskerroksen päälle erityyppisiä lisäpalveluita. Esitystapakerroksella määritellään tiedon esitystapa sovellusten välillä. Sovelluskerros palvelee suoraan loppukäyttäjää.

Seuraavaksi käydään läpi kuvan 2.3 SS7-protokollakerrokset, jotka voidaan peilata OSI-malliin. MTP (Message Transfer Part) huolehtii kahden verkkoelementin välisestä merkinannosta puhelun muodostuksessa. MTP koostuu kolmesta protokollakerroksesta (kuva 2.4). Ensimmäinen kerros määrittelee fyysiset ja elektroniset ominaisuudet. Toinen kerros huolehtii virheettömästä merkinantosanomien kuljetuksesta elementtien välillä. Kerroksen tehtävänä on siis virheen korjaus ja sanomien erotus. Kolmantena on verkkokerros, joka välittää merkinantosanomat elementtien välillä. Esimerkiksi reititys kuuluu tämän kerroksen tehtäviin. /5/



Kuva 2.4 MTP:n protokollakerrokset /1/

Kuvassa 2.3 ’User Parts’ -kerros voi olla TUP (Telephone User Parts), NUP (National User Part) tai ISUP (ISDN User Part). Nämä ovat puhelun käyttäjäosan välitykseen tarkoitetut protokollat, joilla muodostetaan, ylläpidetään ja lopetetaan puhelu hallitusti. Koska tarvitaan myös virtuaalisia yhteyksiä, niin matkapuhelinkeskuksessa on protokolla nimeltä SCCP (Signalling Connection and Control Part). SCCP hallitsee sekä virtuaaliset että yhteydettömän (puheluun liittymätön) ja yhteydellisen merkinannon. Protokollaa käytetään sekä merkinantoyhteyksien muodostamiseen että merkinantoyhteyksien toimivuuden tarkistamiseen rinnakkaisesti ’User Part’ -kerroksen kanssa. /5/

GSM-verkon merkinanto on monimutkaisempaa kuin kiinteässä verkossa. Pääasiassa ero johtuu matkapuhelinten liikkuvuudesta ja siihen liittyvistä

merkinantosanomista. GSM-verkkoon on tästä syystä määritelty kolme seuraavaa protokollakerrosta: BSSAP (Base Station Subsystem Application Part), MAP (Mobile Application Part) ja TCAP (Transaction Capabilities Application Part). /5/

BSSAP-protokollakerrosta käytetään kun MSC kommunikoi BSC:n ja matkapuhelimen kanssa. Koska matkapuhelin ja MSC kommunikoivat BSC:n välityksellä, tarvitaan virtuaalinen yhteys. Tästä syystä myös SCCP:n palveluita tarvitaan. SCCP:n päällä oleva BSSAP kuljettaa matkapuhelimen ja MSC:n välillä tilaajan tunnistamiseen liittyvän merkinannon. Myös BSC ja MSC lähettävät toisilleen merkinantoa, jonka BSSAP välittää. /5/

MAP-protokollakerrosta käytetään NSS-elementtien väliseen signalointiin. Matkapuhelinosa on siis käytössä GSM-verkon keskusten ja rekistereiden välillä eli esimerkiksi MSC:n ja HLR:n välillä. Kahden MSC:n välinen merkinanto käyttää MAP:ia vain silloin, kun kyse ei ole käyttäjän liikennöinnistä. MAP:n avulla mm. sijaintitiedot päivitetään. MAP:n yhteydessä on elementtien välillä lähetettävä tyypillisesti useita sanomia, joissa on pyyntö tai vastaus ja näihin liittyvä lisämääre. Näitä tapahtumia ohjaa MAP:n alla oleva TCAP-protokollakerros. /5/

2.3 ISUP-signalointi

Tämä työ tehdään ympäristössä, jossa testaus keskittyy ISUP-signalointiin. Tästä syystä tässä luvussa kerrotaan tarkemmin ISUP-signaloinnin historiasta, standardeista ja sanomista.

2.3.1 Signalointistandardit ja niiden historia

Signaloinnin toteuttamiseksi ovat olemassa standardoidut sanomat keskuksien ja tietoliikenneverkon eri osien välillä. Eri puolilla maailmaa on kehitetty omia standardeja kyseisille sanomille. Eri standardien sanomat tekevät saman asian, mutta ovat hieman eri muodossa toisiinsa nähden. Tämä on johtanut siihen, että eri

maiden välisissä puheluissa on pitänyt ottaa huomioon eri signaloinnit. Näiden eroavaisuuksien vuoksi kansainvälinen televiestintäliitto (ITU) suositteli standardiksi kanavakohtaista merkinantoa CAS (Channel Associated Signalling), jossa jokaisella puhekanavalla on oma merkinantokanavansa.

CAS-signalointi todettiin kuitenkin hyvin nopeasti riittämättömäksi, koska se toimi hyvin ainoastaan, kun liikennettä oli vähän. Toinen merkittävä epäkohta CAS-signaloinnissa on, että siinä ei ole mahdollista lähettää signalointiviestejä puhelun aikana. ITU kehitti 80-luvun alkupuolella uuden signalointisysteemin, nimeltä yhteiskanavamerkinanto. Tämän signaloinnin suurin hyöty oli, että signaloinnin ei tarvitse kulkea samaa reittiä kuin puheen. Kyseinen signalointi voidaan lyhentää monella tapaa, CCS7, CCS#7, SS7 tai C7 (Common Channel Signalling System Number 7). Alun perin SS7 koostui kahdesta osasta, MTP:stä ja TUP:sta. MTP:n tehtävä oli hoitaa sanomien siirto signalointiverkossa ja TUP:n tehtävä oli sanomien käsittely.

Käyttöönotettaessa ISDN:ää (Integrated Services Digital Network) tarvittiin joitakin uusia sanomia. ISDN on digitaalinen monipalveluverkko, joka tarjoaa digitaalisia yhteyksiä käyttäjä-verkko -rajapinnan välille. Kyseistä signalointia aloitettiin kutsua nimellä ISDN User Part (ISUP). TUP, NUP ja ISUP hoitavat kaikki samaa asiaa puhelun yhdistämisessä. Ajan kuluessa ja teknologian kehittyessä signaloinnin vaatimukset lisääntyivät. Ymmärrettiin, että TUP:n ja MTP:n yhdistelmä ei yksinään riitä, kun virtuaaliset yhteydet tulivat tarpeellisiksi. Sanomat voivat kulkea signalointipisteestä A pisteeseen B monta eri reittiä. Tällöin sanomat eivät välttämättä tule määränpäähän samassa järjestyksessä kuin niiden alkuperä on. Tästä syystä virtuaalinen yhteys tuli tarpeelliseksi. Virtuaalinen yhteys käyttää yhteysorientoitunutta protokollaa, joka huolehtii sekvenssijärjestyksestä loppupäässä. TUP:n ja MTP:n yhdistelmään tarvitaan lisäystä myös silloin, kun signalointisanoma pitää lähettää monien verkkojen yli. MTP pystyy kuljettamaan sanomaa vain yhden verkon sisällä.

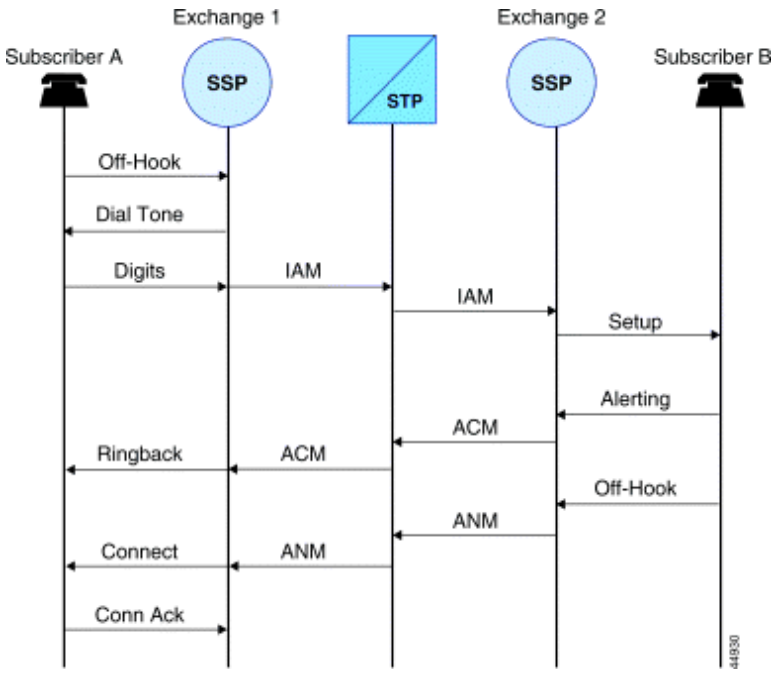
Ratkaisuna näihin ongelmiin kehitettiin uusi protokollakerros MTP:n päälle, jota kutsuttiin SCCP:ksi (Signalling Connection and Control Part). Se huolehtii

virtuaalisista yhteyksistä ja yhteydettömästä signaloinnista. SCCP:n ja TUP:n tehtävät ovat erilaiset, vaikka molemmat käyttävät MTP:n palveluita. SCCP on välikerros, joka toimii linkkinä MTP:stä BSSAP:hen ja MAP:iin sekä toisinpäin.

Nämä edellä mainitut protokollakerrokset hoitavat PSTN-verkon (Public Switched Telephone Network) toiminnan.

2.3.2 Sanomasekvenssi

Signaloinnissa on lukuisia eri tilanteita kuvaavia sanomia. Käydään esimerkkinä läpi ISUP-signaloinnin sanomasekvenssi peruspuhelun osalta (kuva 2.5 ja 2.6).

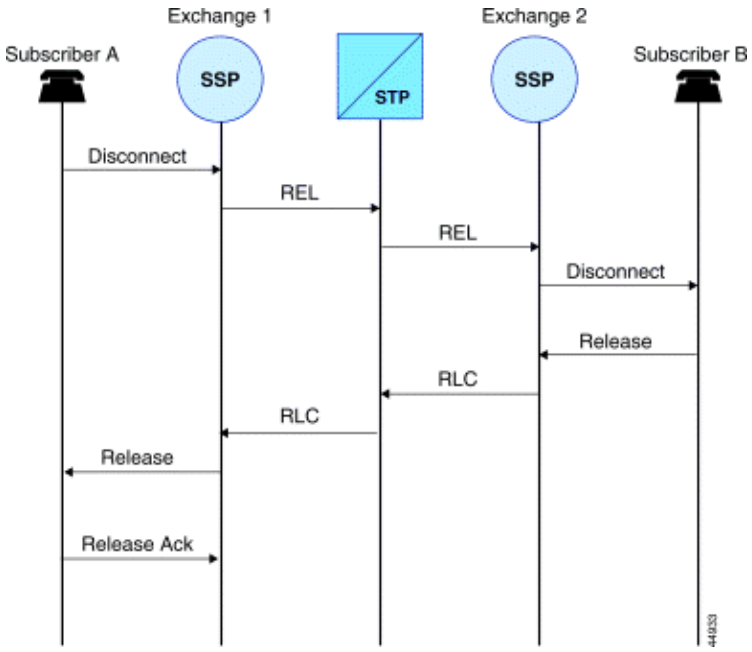


Kuva 2.5 Sanomasekvenssi puhelun muodostamisesta /18/

Sanomasekvenssin vaiheet puhelun muodostamisessa:

1. A-tilaaja (Subscriber A) valitsee B-tilaajan (Subscriber B) numeron.

2. Alkupään keskus (SSP, Exchange 1) lähettää ISUP-signaloinnin IAM-viestin (Initial Address Message) varatakseen tiedonsiirtoa varten tarvittavat resurssit.
3. IAM-viesti välitetään keskuksen (STP) läpi.
4. Loppupään keskus (SSP, Exchange 2) tarkistaa valitun numeron käyttäen reititystaulukkoa ja lähettää ACM-viestin (Address Complete Message), joka kertoo, että B-tilaajan puhelin hälyttää.
5. A-tilaaja kuulee soiton merkkiäänen.
6. B-tilaaja vastaa puhelimeen, ANM-viesti (Answer Message) lähetetään alkupään keskukselle.
7. A-tilaajan päätelaite vastaa lähettämällä ’Conn Ack’ -viestin. Puhelu on puhetilassa.



Kuva 2.6 Sanomasekvenssi puhelun purkamisesta /18/

Sanomasekvenssin vaiheet puhelun purkamisesta:

1. A-tilaaja sulkee puhelimen, keskus 1 lähettää REL-viestin (Release Complete) signalointiverkkoon.

2. REL-viesti välitetään keskuksen läpi.
3. Keskus 2 vastaanottaa REL-viestin ja purkaa resurssit. B-tilaajalle ilmoitetaan, että A on lopettanut puhelun.
4. Keskus 2 lähettää RLC-viestin (Release Complete) A:n keskukselle, jonka johdosta yhdysjohto vapautetaan.
5. A-tilaajan päätelaite vastaa lähettämällä ’Release Ack’ -viestin. Puhelu on purettu onnistuneesti.

3 OHJELMISTON TESTAUS

Tavanomaisessa puhekielessä termillä testaus tarkoitetaan mitä tahansa kokeilemista. Ohjelmistojen testauksen yhteydessä testaus määritellään perinteisesti suunnitelmalliseksi virheiden etsimiseksi, ohjelmaa tai sen osaa suorittamalla /7/. Tässä luvussa käsitellään ohjelmiston kehitysprosessia ja sen elinkaarimalleja lyhyesti. Lisäksi kuvataan testaukseen liittyviä tasoja ja ongelmia sekä testausmenetelmiä ja testauksen dokumentointia.

Ohjelmistotestausta tullaan tarvitsemaan aina ohjelmistotuotannon yhteydessä. Koska ohjelmia tekevät ihmiset, on inhimillistä, että virheitä tapahtuu. Ohjelmistovirheellä tarkoitetaan tilannetta, että ohjelma ei toimi niin kuin sen on suunniteltu toimivan. Virheetöntä ja täydellistä ohjelmaa ei ole olemassakaan /12/. Jotta kriittisimmät virheet saataisiin ohjelmasta karsittua, tarvitaan testausta. Ideaalitilanne olisi testata jokainen ohjelman variaatio. Käytännössä tämä ei kuitenkaan ole mahdollista. Jopa hyvin yksinkertaisella ohjelmalla voi olla satoja tai tuhansia eri tulo- ja lähtöyhdistelmiä. Testitapausten tekeminen näille kaikille mahdollisuuksille ei ole järkevää /10/. Yksi merkittävimmistä rajoittavista tekijöistä tälle on aika. Testaukseen käytettävä aika on oltava tehokasta ja järkevästi suunniteltua, jotta ohjelmasta tulisi laadukas.

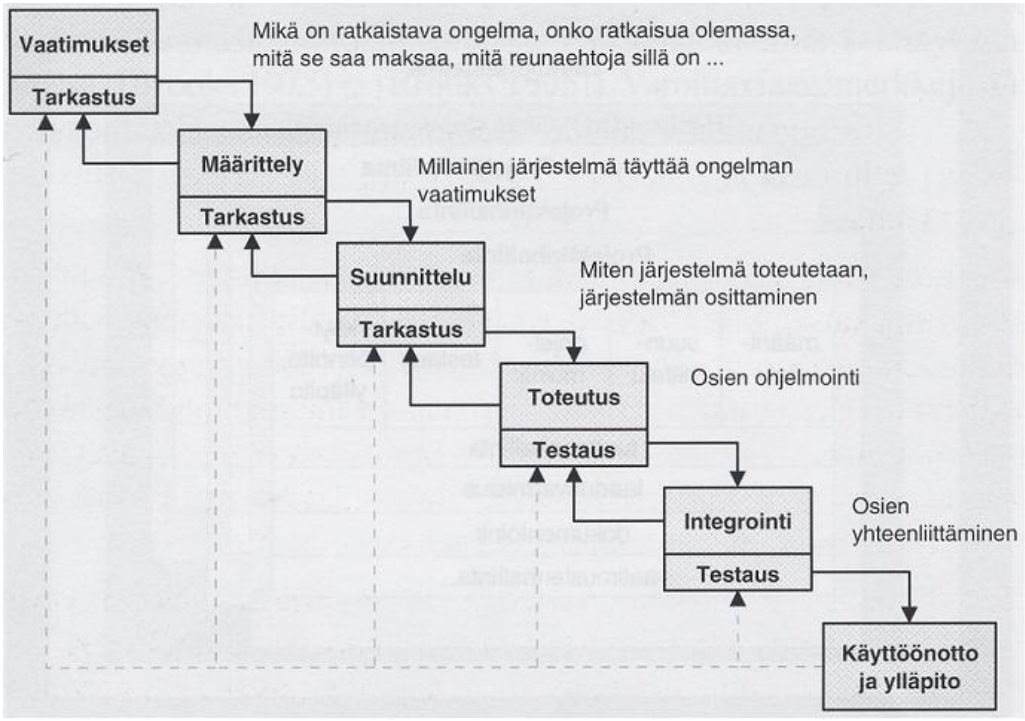
Usein ajatellaan virheellisesti, että testauksella todistetaan ohjelman olevan virheetön. Toimintavarmuuden lisääminen tarkoittaa virheiden löytämistä ja poistamista. Tästä syystä ohjelman testauksella ei todisteta sen toimivuutta, vaan oletetaan ohjelmassa olevan virheitä ja suoritetaan testitapauksia niiden löytämiseksi. /4/

Kit:n (1995) mukaan ohjelmistotestaus voidaan tiivistää kuuteen perusperiaatteeseen:

1 Testaukseen panostaminen määrittelee testausprosessin laadun.
2 Virheen suurenemisen voi ehkäistä käyttämällä testaustekniikoita mahdollisimman aikaisessa vaiheessa.
3 Aika ohjelmistotestauksen työkalukehitykselle on aina ajankohtainen.
4 Oikean ihmisen on vastattava testausprosessin kehittämisestä.
5 Testaus vaatii ammattitaitoisia ja taidokkaita testaajia.
6 Testausympäristöön tarvitaan luova ja positiivinen ilmapiiri. /8/

3.1 Ohjelmiston kehitysprosessi

Ohjelmiston kehitysprosessia voidaan mallintaa eri tavoilla. Tässä aliluvussa on lyhyt kuvaus ja vertailu yleisimmistä prosessimalleista. Ohjelmiston kehitysprosessia kuvataan usein ns. vesiputousmallilla. Siinä ylimpänä on itse idea ohjelmistosta, ja askeleittain päädytään lopulta lopulliseen tuotteeseen. Jokaisen tason jälkeen tarkastellaan, ollaanko valmiita siirtymään seuraavaan vaiheeseen. Vesiputousmallin vaiheet nähdään kuvasta 3.1. Vaiheet ovat: vaatimukset, määrittely, suunnittelu, toteutus, integrointi ja käyttöönotto sekä ylläpito.



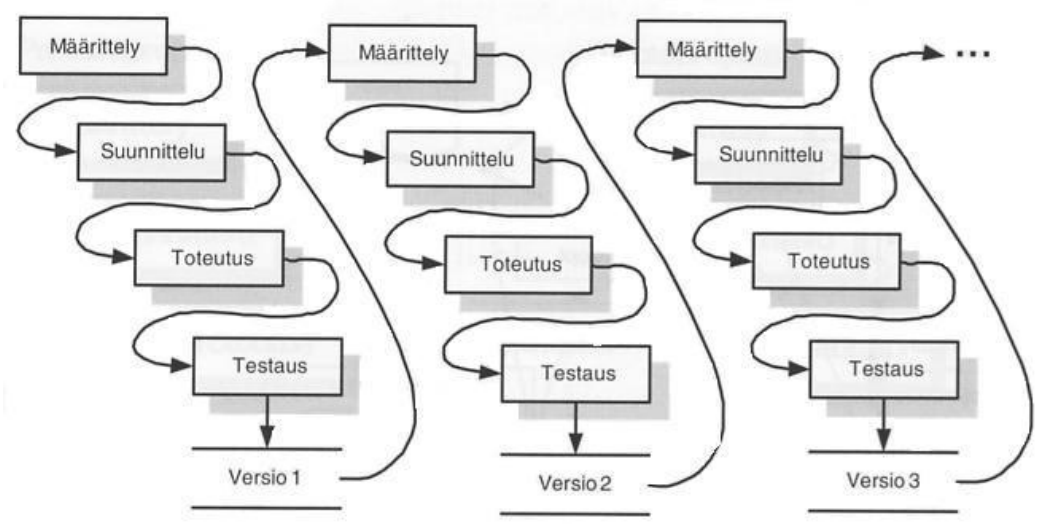
Kuva 3.1 Esimerkki vesiputousmallista /7/

Periaatteessa mallin mukaisesti vaiheiden välillä ei ole päällekkäisyyttä ja takaisinpäin ei voida mennä. Tästä on kuitenkin poikkeuksia. Kuten huomataan, koko kehitysprosessissa on monta vaihetta. Monessa vaiheessa määritellään tuotetta sekä analysoidaan ja suunnitellaan sitä. Tästä syystä edellä mainittujen virheidenkin todennäköisyys näissä vaiheissa lisääntyy. Tämän mallin huonona puolena on testauksen tekeminen ainoastaan prosessin loppupäässä. Jo alkuvaiheessa tullut virhe kulkee melkein koko prosessin läpi, ja näin ollen korjauskustannuksetkin suurenevat.

Käytännössä ohjelmistokehitys ei voi koskaan edetä kirjaimellisesti vesiputousmallin mukaisesti, koska mm. osa vaatimuksista selviää vasta projektin aikana. Tämän lisäksi vaatimukset, lähes poikkeuksetta, muuttuvat ajan mittaan. Vesiputousmallia voidaan kuitenkin pitää todellisen toiminnan mallina, jonka mukaisesti pyritään toimimaan siinä määrin kuin mahdollista. /7/

Vesiputousmallin lisäksi yleisesti käytettyjä kehitysprosessimalleja ovat mm. prototyyppimalli ja erilaiset inkrementaaliset mallit. Prototyyppilähestymistavalla voidaan tarkoittaa lähes mitä tahansa työskentelymallia, jossa jotain tuotteen piirrettä kokeillaan ennen varsinaisen tuotteen rakentamista. Prototyypit soveltuvat erityisesti uuden teknisen ratkaisun vaatiman kokeilun tekemiseen tai etsimään epäselviä asiakasvaatimuksia. Ongelmana prototyyppimallissa voi olla se, että asiakas luulee lähes oikealta näyttävän järjestelmän olevan jo käytännössä valmis, vaikka valtaosa työstä on vielä tekemättä. /7/

Käytännössä useimpien tuotekehityshankkeiden eteneminen tapahtuu syklisesti: järjestelmästä julkaistaan aina tietyin väliajoin uusilla ominaisuuksilla täydennetty versio. Uuden version kehitysprojekti käynnistyy etsimällä edellisestä versiosta saadusta palautteesta tärkeimmät työkohteet. Näiden versioiden lisäksi voidaan joutua toimittamaan lukuisia asiakaskohtaisia versioita, joissa asiakkaiden raportoimia vakavia virheitä on korjailtu. Tuotekehityksen edetessä kuvan 3.2 mukaisesti sarjana pitkäkestoisia vesiputouksia palaute tehtyjen ratkaisuiden toimivuudesta tulee vasta kehityssyklin myöhäisessä vaiheessa. Tähän liittyy useita virheiden korjaamista koskevia ongelmia. /7/



Kuva 3.2 Tuotekehityssyklit /7/

Inkrementaalimalli on muuten samanlainen kuin vesiputousmalli, mutta suunnittelun jälkeiset vaiheet suoritetaan useaan kertaan. Periaatteessa kehitys etenee kuten kuvassa 3.2, mutta yksittäiset syklit eli iteraatiot ovat tyypillisesti hyvin lyhyitä. Jokainen iteraatio sisältää edellisten versioiden toiminnallisuuden lisäksi uusia piirteitä ja ominaisuuksia. Kaikki toimitettavat versiot pohjautuvat samoihin vaatimuksiin, määrittelyyn ja suunnitteluun. /19/ Näin meneteltäessä esimerkiksi tehtyjen kriittisten suunnitteluratkaisujen käyttökelpoisuutta päästään testaamaan jo projektin alkuvaiheessa. Lisäksi vesiputousmallin pitkäkestoisuudesta johtuvaa ongelmaa pystytään ainakin periaatteessa ratkaisemaan. /7/

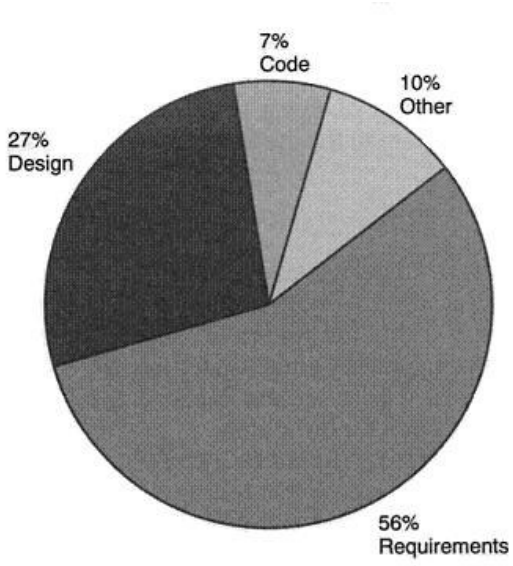
Yksi ohjelmiston kehitysprosessia kuvaava malli on spiraali. Tämän mallin ideana on, että kaikkea ei määritellä heti alussa yksityiskohtaisesti. Ensin määritellään tärkeimmät ominaisuudet ja kokeillaan niiden tekemistä ja palautteen kautta jatketaan eteenpäin. Näin edetään, kunnes koko ohjelmisto on valmis. Jokainen spiraalin kierros sisältää samat vaiheet: kohteiden määrittely, vaihtoehdot ja rajoitukset, riskien hallinta, kyseisen tason kehitys ja testaus, seuraavan tason suunnittelu ja viimeisenä lähestymistapa seuraavalle tasolle. Testaajan kannalta spiraalimalli on parempi kuin vesiputousmalli, koska spiraalimallin eri vaiheiden johdosta testaus voidaan aloittaa tuotekehityksen aikaisessa vaiheessa.

3.2 Ohjelmistovirheet ja niiden kustannukset

Ohjelmistovirhe on ohjelmiston kehitysvaiheessa ohjelmistoon jäänyt piilevä virhe. Ohjelmistovirheitä ovat mm. vaatimusmäärittelyn virheet, suunnitteluvirheet ja ohjelmointivirheet. Nykyaikaiset ohjelmointiympäristöt kääntäjineen tunnistavat syntaksivirheet sekä karkeimmat muistiosoituksien virheet automaattisesti. Painopiste on näin siirtynyt vaatimusmäärittely- ja suunnitteluvaiheita kohti. Käytännössä täysin virheetöntä ohjelmaa on mahdoton valmistaa, ja ohjelmistoon jää väistämättä piileviä virheitä. /20/

Suotuisissa olosuhteissa ohjelmistovirhe etenee virhetilanteeksi. Etenemisen edellytyksenä toimii heräte. Herätteitä voi olla käytännössä kahta tyyppiä: vääriä syötteitä tai vääriä ajoituksia. Heräte voi olla ihmislähtöinen (esimerkiksi käynnistetään väärä ohjelma epäsuotuisaan aikaan), ympäristölähtöinen (esimerkiksi rikkoutunut komponentti tuottaa virheellisiä tuloksia) tai ohjelmiston sisäinen (esimerkiksi virhe taulukon tilavarauksessa aiheuttaa taulukon ulkopuoliseen muistipaikkaan kirjoittamisen). /20/

Pattonin (2005) mukaan suurin syy ohjelmistovirheisiin on määrittelydokumentti. Tämä johtuu taas usein siitä, että koko dokumenttia ei ole edes kirjoitettu tai sitä ei ole kirjoitettu huolellisesti. Toiseksi suurin syy ohjelmistovirheisiin on Pattonin mukaan suunnitteluvaihe, jolloin ohjelmoijat laativat suunnitelman ohjelmiston tekemisestä. Syynä suunnitteluvaiheesta aiheutuviin virheisiin ovat hätiköinti, muutosten teko ja kommunikaation puuttuminen muun työryhmän kanssa. Vasta kolmantena, suurimpana syynä ohjelmistovirheisiin Patton pitää itse koodia ja siinä olevia virheitä /2/. Kit kuvaa virheiden jakautumista kuvassa 3.3. Kuvan mukaan suurin osa (56 %) virheistä johtuu vaatimuksista. Seuraavaksi eniten virheitä aiheuttaa suunnitelma (27 %). Vähiten virheitä aiheuttaa ohjelmoitu koodi (7 %). Loput virheistä johtuvat muista syistä. /8/

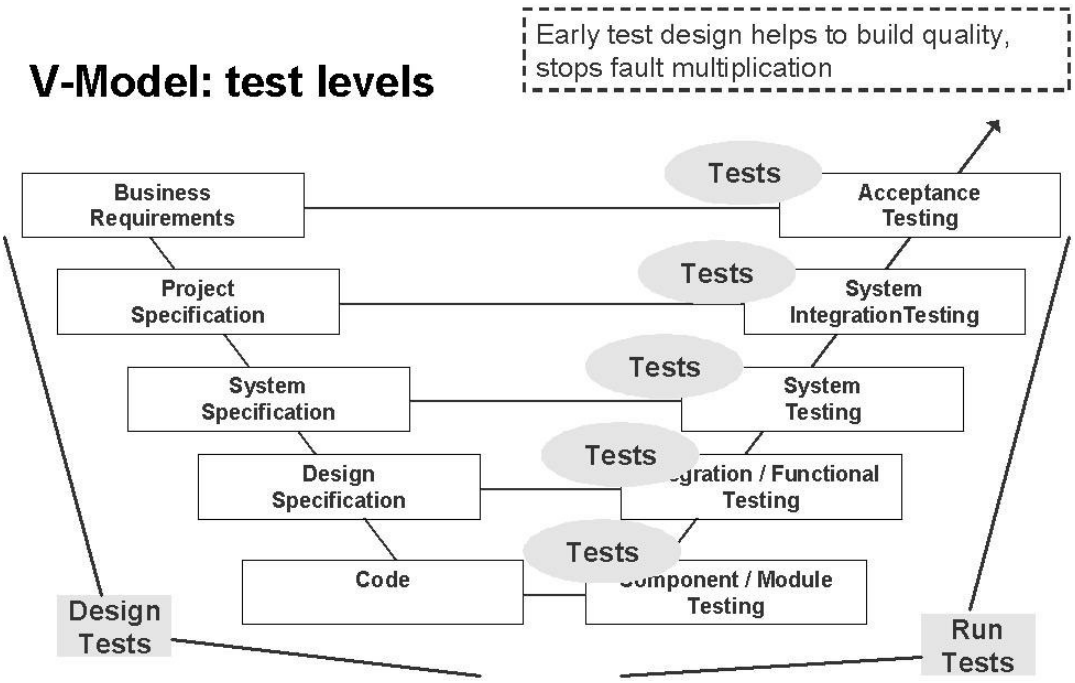


Kuva 3.3 Ohjelmistovirheiden jakauma /8/

Kun ohjelmistovirhe huomataan aikaisessa vaiheessa ohjelmiston tekemistä, siitä johtuvat kustannukset ovat pienet ja virhe on helposti korjattavissa. Mikäli virhe taas huomataan ohjelmiston ohjelmointi- tai testausvaiheessa, kustannukset voivat kasvaa jopa kymmenkertaisiksi, puhumattakaan kustannuksista, jotka aiheutuvat, kun virhe huomataan vasta asiakkaalla /11/. Lopullisessa toimintaympäristössä esiintyvästä kriittisestä ohjelmistovirheestä voidaan joutua maksamaan jopa ihmishenkiä.

3.3 Testauksen vaiheet ohjelmiston kehitysprosessissa

Testaus tapahtuu yleensä monella tasolla ns. V-mallin mukaisesti. Kuvan 3.4 V-mallissa testaus jaetaan moduulitestaukseen (yksikkötestaus), toimintotestaukseen (integrointitestaus) ja järjestelmätestaukseen. Järjestelmätestausta voi seurata erillinen kenttätestaus ja/tai hyväksymistestaus. Moduulitestauksessa etsitään virheitä yksittäisistä moduuleista, integrointitestauksessa moduulien yhteistoiminnasta ja järjestelmätestauksessa koko järjestelmän toiminnoista ja suorituskyvyistä. V-mallin mukaisesti testauksen suunnittelu tapahtuu testaustasoa vastaavalla suunnittelutasolla. Kuvassa 3.4 ohjelmakoodi testataan moduulitestauksessa, arkkitehtuurisuunnittelu toimintotestauksessa, järjestelmäsuunnittelu järjestelmätestauksessa jne.



Kuva 3.4 Testauksen V-malli /15/

Mitä korkeammalla V-mallin testaustasolla ollaan, sitä kalliimmaksi virheiden korjaus tulee. Virheiden korjaus voi myös aiheuttaa uusia virheitä. Jos esimerkiksi järjestelmätestauksessa havaittu virhe korjataan, voi korjaus aiheuttaa muutoksia muihinkin moduuleihin. Siltä varalta, että jokin muutostarve jää huomaamatta, myös muut moduulit pitäisi testata ja lopuksi vielä suorittaa järjestelmätestaus uudelleen. Tällaista uudelleentestausta kutsutaan regressiotestaukseksi ja sen suorittaminen toistuvasti vie paljon aikaa sekä tulee kalliiksi, ellei testausta ole automatisoitu. /7/ Testauksen automatisoinnista on kerrottu lisää luvussa 4. Regressiotestauksen suunnittelua helpottaa matriisi, jossa testitapaukset on linkitetty toimintoihin. Näin ollen matriisista nähdään helposti, mitkä testitapaukset tulee suorittaa, kun tiettyä toimintoa on muutettu. Matriisia pitää muistaa päivittää sitä mukaa, kuin toimintoja tulee lisää tai niitä muokataan. /9/

Yleisesti käytetyt termit, joilla kuvataan testauksen lähestymistapaa, ovat musta- ja valkolaatikkotestaus (Black-Box Testing, White-Box Testing). Mustalaatikkotestauksessa testaaja tietää ainoastaan, mitä ohjelman pitäisi tehdä, ei sitä, miten. Kun siis ohjelmaa testataan tietynlaisella syötteellä, niin lopputuloksen

pitää olla tietynlainen. Valkolaatikkotestauksessa tiedetään ja otetaan huomioon, mitä laatikko eli ohjelma sisältää. Toimintotestaus on mustalaatikkotestausta ja moduulitestaus on valkolaatikkotestausta /2/. Käytettäessä mustalaatikkotestausta, ohjelma ajatellaan mustana laatikkona. Tarkoituksena on jättää ohjelman sisäinen rakenne huomioimatta ja keskittyä olosuhteisiin, jolloin ohjelma ei toimi halutulla tavalla. Toisessa testausstrategiassa valkolaatikkotestauksessa niin sanotusti nähdään ohjelman sisään. /4/

Näiden lisäksi on vielä olemassa harvinaisempi harmaalaatikkotestaus, jossa käytetään hyväksi tietoa ohjelman toteutusperiaatteista. Siirryttäessä V-mallissa alimmalta tasolta ylöspäin testauksen luonne muuttuu valkolaatikkotestauksesta yhä enemmän mustalaatikkotestaukseksi. /7/

Pattonin mukaan yksi tapa kuvata eri testaustapoja on jako staattiseen ja dynaamiseen testaukseen. Staattisella testauksella tarkoitetaan ei-käynnissä olevan prosessin testausta. Dynaamisessa testauksessa testattavaa ohjelmaa suoritetaan tai ohjelma on käynnissä. Havainnollinen esimerkki kyseisistä testaustavoista on, kun tarkistetaan käytettyä autoa. Ensin suoritetaan staattista testausta eli potkitaan renkaita, tarkistetaan maalipintaa ja katsellaan autoa sisältä. Tämän jälkeen siirrytään dynaamiseen testaukseen käynnistämällä auton moottori, kuuntelemalla moottorin ääntä ja ajamalla koeajo. Ohjelmistotestauksessa esimerkiksi määrittelydokumentin tarkastelu on staattista testausta, koska dokumentti ei ole suoritettava ohjelma. /2/

3.4 Ohjelmistotestaajan tehtävä

Ohjelmistotestaajan työssä pitää tarkastella ohjelmiston toimivuuden kannalta kriittisimpiä asioita ja testata niitä. Ohjelmistotestaaja ei testaa todistaakseen, että ohjelma toimii, vaan tehtävänä on löytää virheet ohjelmasta. Koska on todennäköistä, että virheetöntä ohjelmaa ei ole, on hyvä ajatella, että virheen löytyessä testi on onnistunut. Virheiden etsimisessä testaajan pitää miettiä, miten virheet löytyvät mahdollisimman aikaisessa vaiheessa, nopeasti ja tehokkaasti.

Pattonin mukaan ohjelmistotestaajalle voidaan määritellä kolme päämäärää: virheiden löytäminen, virheiden löytäminen niin aikaisessa vaiheessa kuin mahdollista ja huolehtiminen siitä, että virheet tulevat korjatuksi.

Testaukseen liittyvät työvaiheet ovat suunnittelu (testisuunnitelma, testitapaukset), testiympäristön luonti, testin suorittaminen, tulosten tarkastelu ja raportointi. Näihin työvaiheisiin ja niihin liittyvään virheiden jäljitykseen ja korjaukseen kuluu tyypillisesti yli puolet ohjelmistoprojektin resursseista. Tästä syystä testauksen läpivientiin kannattaa kiinnittää erityistä huomiota. /7/

Myersin (2004) mukaan yksi vaikeimmista asioista testauksessa on määritellä, koska on testattu tarpeeksi. Kaksi yleistä kriteeriä testauksen lopettamiselle on: lopeta silloin, kun aikataulu niin määrää ja toinen, lopeta, kun kaikki testitapaukset on ajettu ja yhtään virhettä ei löytynyt. Ensimmäinen näistä kriteereistä on kuitenkin hyödytön, koska sen pystyy täyttämään testaamatta ollenkaan. Jälkimmäinen kriteereistä taas ei takaa laadukasta testausta, vaan ennemminkin kannustaa ajamaan helppoja varmasti läpi meneviä tapauksia /4/. Tuotekehitystyössä testauksen lopettaminen on usein kompromissi tuotteessa olevien virheiden aiheuttamien kustannusten ja markkinoilta myöhästymisen aiheuttaman tuoton menetyksen välillä /7/. Tähän liittyy myös riskien hallinta, koska on riski jättää jokin asia testaamatta, ja näin ollen ohjelmaan voi jäädä virhe. Toisaalta, koska kaikkea ei voida testata, testaajan pitää pystyä tekemään järkeviä päätöksiä testattavien asioiden ja ajankäytön suhteen. Tehokkaalla ohjelmistotestaajalla on hyvä olla tietämys myös ohjelmiston koko kehitysprosessista. /2/

Kun testaajalla on tiedossa minkälainen tulos pitää tietyllä syötteellä tulla, niin seuraavaksi pitää määritellä testitapaukset. Testitapauksilla määritellään syötteet joilla testataan ja testausprosessi, jota testissä noudatetaan. Yksi testaajan tärkeimmistä tehtävistä on testitapausten valinta. Pitää valita kattava määrä sopivia testitapauksia.

Edellisessä aliluvussa 3.3 esiteltyjä musta- ja valkolaatikkotestaustapoja voidaan ajatella myös testitapausten valinnan lähestymistavaksi. Mustalaatikkotestauksessa testitapaukset valitaan testattavan ohjelman määrittelydokumenttien perusteella tutustumatta ohjelman toteutukseen. Valkolaatikkotestauksessa testitapausten valinnassa käytetään hyväksi tietoa ohjelman toteutuksesta. /7/

Testauksen luonne voi olla helposti läpäistävä perustestaus tai hankala rajoja koetteleva testaus. Pattonin mukaan suositeltava tapa on testata ensin helpot testitapaukset ja vasta sen jälkeen vaikeammin läpäistävät, koska ohjelman perustoimivuus on tärkeää todeta. Testaajan tehtävänä on kuitenkin löytää virheitä, joten perustoimivuuden toteamisen jälkeen tarkoituksena on koetella ohjelmaa hankalilla ja ääritilanteita testaavilla tapauksilla.

Toisen testaajan työ voi edistää virheiden löytämistä. Tosiasia on, että ihmiset huomioivat eri asioita. Jos sama ohjelmisto testattaisiin monen testaajan toimesta, he löytäisivät varmasti eri virheitä. Tähän vaikuttaa paljon jo pelkästään testitapausten valinta. Toisen testaajan työskentelyn seuraaminen voi tuoda uusia lähestymistapoja testaukseen ja näin ollen parantaa tulosta. /2/

3.5 Testauksen dokumentointi

Testaajat suunnittelevat ja dokumentoivat heidän työnsä. Dokumentointeja ovat mm. testaussuunnitelma, testitapausten kuvaus ja tulosten raportointi. Testaussuunnitelmasta selviää mm. mitä testejä tehdään ja milloin, miten ne järjestetään ja millaisia lopputuloksia odotetaan. /7/ Testitapaukset kertovat yksityiskohtaisemmin testattavat asiat sekä testivaiheet, joilla testitapaus käydään läpi. Raporttiin listataan löytyneet virheet ja läpimenneet testitapaukset.

Testausdokumentaatiota voi syntyä varsin paljon jos ajatellaan, että aliluvussa 3.3 esitetyn V-mallin mukaisesti jokaisesta testaustasosta on oma dokumentti ja jokaisen tason alla olevista testitapauksista omansa. Dokumentaation määrä riippuu toki projektin suuruudesta.

Pattonin mukaan testisuunnitelma on suunnittelun sivutuote, koska suunnittelu on tärkeämpää, kuin suunnittelun tuloksena saatu dokumentti /2/. Tämän paikkansapitävyyttä todistaa suunnittelun jälkeisten prosessien epäonnistuminen, mikäli esimerkiksi testitapaukset suunnittelee huonosti. Testisuunnitelman on tarkoitus toimia kommunikaatiovälineenä muun testausryhmän kanssa. Suunnitelmasta näkee helposti suunnittelun tulokset, mitä testataan ja miten testataan, sekä mitkä ovat odotetut tulokset. Testauksen suunnitteluprosessissa yksi taso alaspäin on testitapausten kirjoittaminen. Testitapausten kuvauksen tulisi olla mahdollisimman informatiivinen, koska testitapauksia ajetaan aina uudestaan.

4 TESTAUKSEN JA TULOSTEN ANALYSOINNIN AUTOMATISOINTI

Tämän luvun tarkoituksena on käsitellä automatisoinnin tarkoitusta ohjelmistotestauksessa. Testauksen automatisointiin sisältyy kaikkiaan kolme osa-aluetta: testauksen suoritus, testitulosten analysointi ja testauksen raportointi. Tässä luvussa käydään läpi lyhyesti kaksi ensimmäistä osa-aluetta. Lisäksi tarkastellaan myös automatisoinnissa huomioon otettavia asioita sekä ongelmia.

4.1 Automatisoinnin tarkoitus

Testitapausten valinta ei ole helppo tehtävä. On oltava varma siitä, että testijoukko kattaa ohjelmiston testauksen laadukkaasti. Tähän pyrittäessä on otettava huomioon käytettävissä oleva aika sekä resurssit. Ongelmatilanne syntyy, kun testausta tarvitaan lisää, mutta aika ei riitä. Tehokkaana ratkaisuna tähän ongelmaan on työkalujen kehitys ja käyttö. Työkalut helpottavat ja nopeuttavat testaustyötä, mutta eivät korvaa testaajaa. Automatisointi ei vähennä resursseja, koska osaamista tarvitaan myös automatisointiohjelman käytössä ja testitapausjoukon ylläpidossa.

Testauksen automatisointi aloitetaan yleensä testisuorituksen automatisoinnista. Tarkoituksena on suorittaa automatisointiohjelmalla samat toimenpiteet, mitä manuaalisesti suoritettaisiin. Testitapauksen suorituksen jälkeen tuloksia verrataan odotettuihin tuloksiin. Tulosten analysointi voi viedä tapauksesta riippuen yllättävän paljon aikaa. Mikäli testitapauksen kannalta on mahdollista, analysointi kannattaa myös automatisoida siihen suunnitellulla työkalulla. /2/

4.2 Automatisoinnissa huomioitavaa

Ohjelmisto muuttuu usein ja automatisoituja testitapauksia on muutettava sen myötä. Automatisointiohjelmalla tehty ohjelmakoodi pitää olla mahdollisimman dynaamista ja helposti muokattavissa olevaa /2/. Dynaamisuudella tarkoitetaan ohjelmakoodin yhteydessä esimerkiksi muuttujien dynaamisuutta; muuttujat voivat saada ohjelman suorituksen aikana eri arvoja. Koska automatisointiprosessiin kuluu aikaa, on hyvä muistaa, että se on työskentelytavan kehittämistä ja näin ollen verrattavissa muuhun kehittämiseen.

Koskaan ei saisi liikaa luottaa automatisoituun testaukseen. Mikään automatisointiohjelma ei voi korvata ihmisen ”tunnetta” siitä, että kaikki ei mennyt oikein /2/. Automatisoinnin suhteen on oltava kriittinen, koska epäluotettavalla automatisoinnilla saadaan aikaan enemmän harmia, kuin hyötyä.

4.3 Automatisoinnin ongelmat

Automatisoinnissa on otettava huomioon myös sen tuomat ongelmat. Jotta ongelmat eivät tulisi yllätyksenä automatisointia tehdessä, on hyvä selvittää ne etukäteen. Seuraavissa kappaleissa on poimittu tärkeimpiä huomioon otettavia asioita automatisointia tehdessä.

Odotukset siitä, että automatisointi ratkaisee kaikki ongelmat mm. ajan- ja resurssien suhteen, ovat epärealistisia. Todennäköisesti, jos automatisoinnilta

odotetaan liikaa, lopputulos ei vastaa odotuksia, vaikka se olisi hyvin toteutettu. Myös odotukset siitä, että automatisoinnin jälkeen löydetään enemmän virheitä, eivät aivan pidä paikkaansa. On hyvin epätodennäköistä, että testin uudelleen suorittaminen samassa laitteistossa ja samassa testiympäristössä, toisi esiin enemmän virheitä. Automaattiseen testaukseen tarkoitetut työkalut ovat uudelleensuoritusta varten. Näin ollen automatisointi on perusteltua esimerkiksi regressiotestauksessa. Odotusten ohella myös automatisoitavat testitapaukset tulee olla kunnossa. Huonosti suunniteltua ja dokumentoitua testausta ei kannata lähteä automatisoimaan. /13/

Automatisoitu testaus vaatii ylläpitoa. Koska ohjelmistoa muutetaan usein, on myös ohjelmistoa testaavia testitapauksia muokattava. Mikäli testaus on automatisoitu muutostilanteessa, niin myös automatisointia tulee muuttaa. Liian usein valitaan helpompi ja halvempi ratkaisu: ajetaan testitapaukset ohjelmaan tulleen muutoksen jälkeen manuaalisesti, koska automatisoinnin päivittäminen veisi liikaa aikaa ja resursseja. /13/

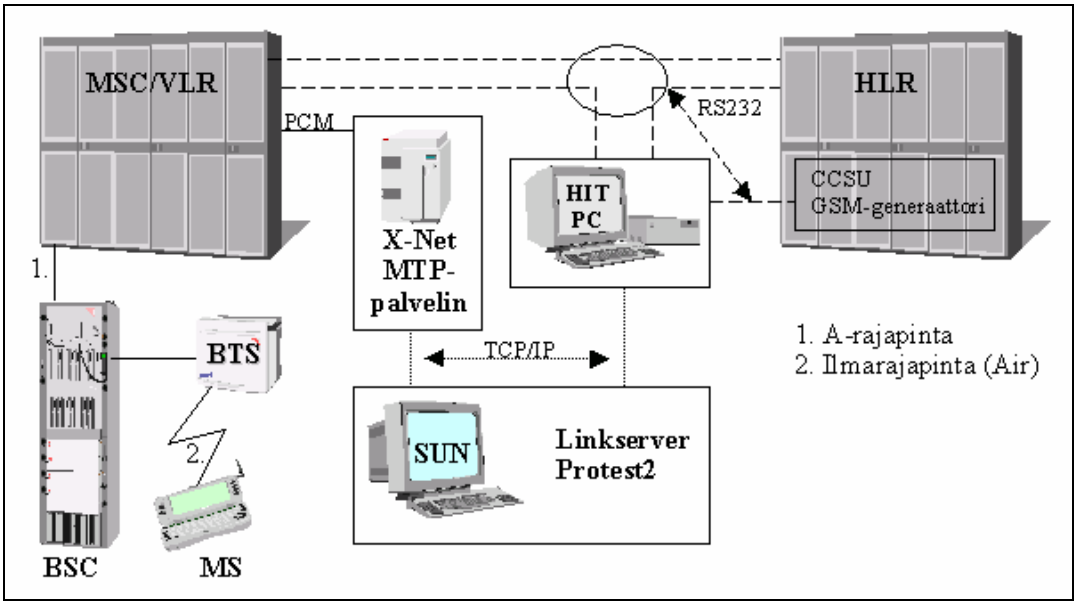
5 SIGNALOINNIN TOIMINTOTESTAUKSEN AUTOMATISOINNIN SUUNNITELMA JA TOTEUTUS

Tämän luvun tarkoituksena on perehtyä signaloinnin toimintotestaukseen. Lisäksi tarkastellaan signaloinnin testausympäristöä ja ympäristöön sisältyviä elementtejä. Tarkoituksena on myös havainnollistaa esimerkin avulla signalointiviestin kulkua eri elementtien välillä. Signaloinnin toimintotestauksesta tehdään kartoitus ja sen pohjalta toteutetaan automatisointisuunnitelma sekä valittujen regressiotestitapausten analysoinnin automatisointi.

5.1 Testausympäristö

Signalointia ja signalointiin liittyviä ominaisuuksia testataan laboratorioympäristössä. Kuvassa 5.1 on esitetty testausympäristön periaatteellinen

kaaviokuva. Testauslaitteisto sisältää matkapuhelinkeskuksen (MSC), kotipaikkarekisterin (HLR), vierailijarekisterin (VLR), tukiasemaohjaimen (BSC), tukiaseman (BTS) ja matkapuhelimia (MS). Lisäksi oikeasta ympäristöstä poiketen, käytetään signalointiliikennesimulaattoria (Protest) ja GSM-generaattoria (GSM-generator). GSM-generaattori on ohjelmalaajennus, joka voidaan ladata HLR:n signalointiyksikköön (CCSU, Common Channel Signalling Unit). Generaattorilla simuloidaan tukiasemaa ja sen alueella sijaitsevia matkapuhelimia. Generaattorilla voidaan peruspuheluiden lisäksi simuloida samoja toiminnallisuuksia, kuin oikeilla matkapuhelimilla. GSM-generaattori hoitaa myös tilaajatietojen päivittämisen VLR:ään ja sijainnin päivitykset. Protest- ja TTCN-ohjelmat (Testing and Test Control Notation) toimivat Unix-pohjaisella SUN-palvelimella. /14/



Kuva 5.1 Testausympäristö /15/

ISDN-verkon toimintaa simuloiva Protest-ohjelma on tarkoitettu SS7:n puhelin- ja ISDN-käyttäjäosien (TUP, ISUP, IUP) testaamiseen. Signalointiliikennettä ohjataan MTP3-palvelimen (Message Transfer Part level 3) ja unix-ympäristössä toimivan Linkserver-ohjelman välityksellä. Windows-käyttöjärjestelmissä toimivat MTP3- ja

LAPD-palvelimet (Link Access Procedure on the D-channel) toimivat sovittimina matkapuhelinkeskuksesta tulevan PCM-pohjaisen (Pulse Code Modulation) johtomerkinannon ja Protest-ohjelman käyttämän TCP/IP-protokollan (Transmission Control Protocol/Internet Protocol) välillä. Palvelimet muodostuvat PC:n (Personal Computer) lisäkortista ja sitä varten kehitetystä ohjelmistosta /14/. Vastaanottaessaan sanoman simulaattorilta Linkserver lisää asiaankuuluvat protokollaosat sanomaan, ennen kuin lähettää sen eteenpäin MTP3- tai LAPD-palvelimelle. Sama tapahtuu myös toisinpäin, kun sanoma tulee Linkserver:lle, niin palvelin poistaa nämä tiedot.

HIT (Holistic Integration Tester) on Nokia Networks:n suunnittelema ja toteuttama PC-pohjainen Windows-ympäristössä toimiva ohjelma, jolla ohjataan matkapuhelinkeskuksen eri toimintoja. Ohjelmalla ajetaan tapauskohtaisia makroja, jotka ohjaavat GSM-generaattoria, protest-istuntoja ja MML-yhteyksiä (Man-Machine Language). Makro on kokoelma toimenpiteitä tallennettuna toimintosarjaksi. Toimintosarja vastaa komentoja jotka esimerkiksi manuaalisesti syötettäisi näppäimistöllä. MML on tekstipohjainen standardoitu ohjauskieli, joka on suunniteltu helpottamaan järjestelmän suoraa käyttäjähallintaa. HIT on TCP/IP- tai RS232-yhteydessä (standardoitu liitäntätapa) matkapuhelinkeskukseen sekä Protest-simulaattoriin. Testausohjelmaa käytetään myös ohjelmalohkojen välisen sanomaliikenteen keräämiseen matkapuhelinkeskuksen eri yksiköistä. /14/

IDA2 (Integrated DX200 Analyzer) on myös Nokia Networks:n kehittämä ohjelma, jota käytetään GSM-verkkoelementtien testaustulosten analysoimiseen. IDA2:lla voidaan analysoida monitorointitiedostoja, esimerkiksi ohjelmalohkojen välisiä sanomia ja verkkosanomia. IDA2:ta käytetään Windows-ympäristössä. IDA2:ssa on mahdollisuus käyttää automaattista analysointia (IDA2-AA, Automated Analysis), jolloin etukäteen tehtyjen referenssitulosten perusteella sanomamonitoroinneista voidaan tarkistaa esim. sanomajärjestys ja sanomien kenttien arvoja. /14/

5.2 Signaloinnin toimintotestauksen kuvaus

Signaloinnin toimintotestausta havainnollistetaan tarkastelemalla signalointiviestien kulkua testauslaboratorioympäristössä. Eri signalointien (ISUP, TUP, IUP, PBX, BICC) sisäiset arkkitehtuurit eivät paljon eroa toisistaan testausympäristössä. Kuvassa 5.2 on kuvattu MSC, X-net PC ja SUN-palvelin. Kuvalla havainnollistetaan IAM-viestin kulkua transit-puhelun testaamisessa Protest-testerillä. Transit-puhelulla tarkoitetaan keskuksen kautta kulkevaa puhelua, joka ei ala eikä pääty kyseisen keskuksen tilaajalle. Eri signalointien ohjelmalohkot eroavat toisistaan. Esimerkkitapauksena käsitellään ISUP signalointia ja siinä olevaa SI7-ohjelmalohkoa. Seuraavassa taulukossa 5.1 luetellaan muut signalointiohjelmalohkot:

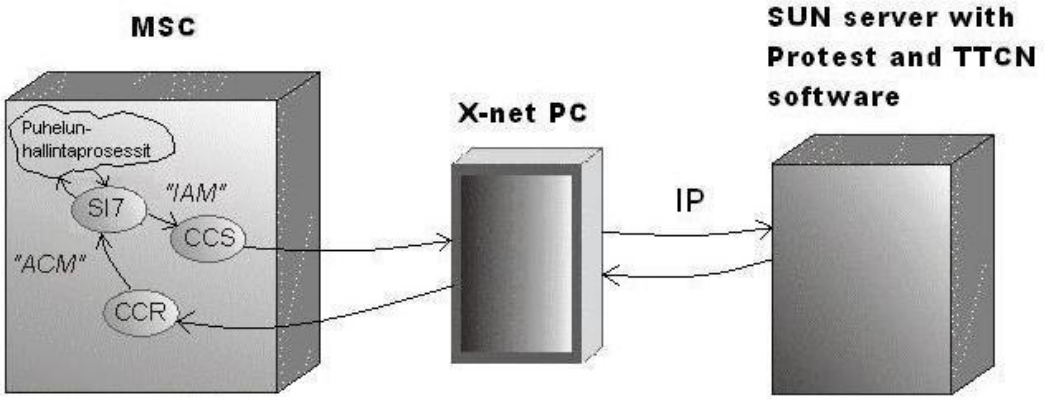
Taulukko 5.1 Signalointiohjelmalohkot

Signalointi	**Signalointiohjelmalohko**
ISUP	SI7
TUP China	SI5
TUP Standardi	SI3
IUP	SI4
PBX DSS1	SI1
BICC	5C9
CAS	SI9

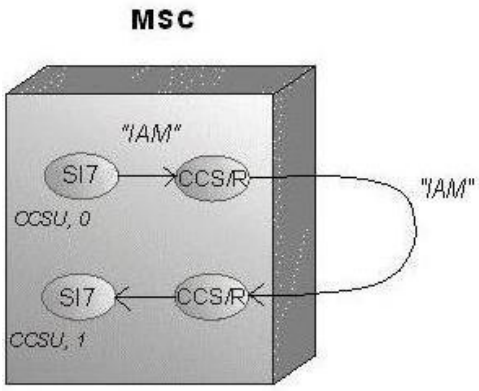
Esimerkkitapauksessa (kuva 5.2) ISUP-signaloinnin IAM-viesti kulkee MSC:n signalointiohjelmalohkosta CCSEND-ohjelmalohkon kautta ulos keskuksesta. Tämän jälkeen viesti kulkee X-net PC:n kautta TCP/IP-yhteydellä SUN-serverille, jossa ovat Protest- ja TTCN-ohjelmistot. Keskukselle tultaessa viesti kulkee CCRECE-ohjelmalohkon kautta SI7-ohjelmalohkolle.

Kuvassa 5.3 kuvataan viestin kulku silmukka-tapauksessa. Saman keskuksen eri CCSU-yksiköt simuloivat puhelun kulkua kahden keskuksen välillä. Protokolla-

analysaattorilla on mahdollista tarkastella ulkorajapinnassa tapahtuvaa liikennettä, mutta tässä testausympäristössä, tällä hetkellä olevilla työkaluilla, protokolla-analysaattori ei tuo lisäarvoa, joten sitä ei käytetä. Signaloinnissa käytetään IDA2-AA työkalua, jolla tarkastellaan keskuksen sisältä ohjelmalohkojen välisiä sanomia.



Kuva 5.2 IAM-viestin kulku transit-tapauksessa



Kuva 5.3 IAM-viestin kulku silmukka-tapauksessa

5.3 Toimintotestauksen nykytilanne

Tätä tutkintotyötä tehtäessä, signaloinnin ulkoiset rajapinnat testataan tarvittaessa TTCN- tai Protest-testerillä. BICC-signaloinnin yhteydessä voidaan käyttää TTCN-testeriä ja muissa signaloinneissa (ISUP, TUP, IUP, PBX) Protest-testeriä. CAS-signaloinnille on oma Ameritec-testeri. TTCN2:lla voidaan testata BICC- ja ISUP-signalointeja. Ulkoista rajapintaa testattaessa varmistutaan siitä, että viesti tulee oikein testerille ja lähtee sieltä oikein takaisin keskukselle. Keskuksen sisältä on myös mahdollista tarkastella signalointiohjelmalohkolta lähtevää ja sinne tulevaa viestiä. Signaloinnin toimintotestauksen tuloksia tarkastellaan - kartoituksen perusteella - sekä ulkoisista rajapinnoista että keskuksen sisältä. Lisäksi tuloksia katsellaan myös verkkoelementtien sisäisistä lokitiedoista ja konfiguraatioon liittyvistä tilatiedoista.

Aiemmin (toukokuu 2005) on suunniteltu automatisoinnin pääpainoa TTCN-testereille. Siirtyminen TTCN2:sta TTCN3:seen ja sen käyttöönotto BICC- ja ISUP-signalointien osalta toisi mukanaan automaation, joka puolestaan nopeuttaa testausta. Tämän muutoksen toteutumiseen menee kuitenkin niin kauan aikaa, että sen varaan ei automatisointia voida laskea.

CCSEND- ja CCRECE-ohjelmalohkojen toiminnallisuuksista kuvataan tässä kappaleessa lyhyesti ainoastaan MTP-osan lisäys ja poisto. Viestin lähtiessä keskukselta ulos CCSEND lisää sanomaan MTP-osan, jonka X-net PC poistaa, ennen kuin välittää viestin Protest-testerille. Samoin viestin tullessa testeriltä keskukselle päin, X-net lisää MTP-osan, jonka keskuksella oleva CCRECE poistaa. Jos oletetaan, että CCSEND- ja CCRECE-ohjelmalohkot sekä X-net PC eivät muuta itse viestin sisältöä keskuksen ja testerin välillä, niin voidaan monitoroida ainoastaan keskuksen sisältä sanomien kulkua ja analysoida tuloksia (monitorointeja) IDA2-AA ohjelmalla.

5.4 Automatisoinnin kehitysprojekti

Tämä tutkintotyö on tehty osana isompaa automatisointiprojektia. Yrityksen eri jaostoissa kartoitettiin tarvetta testitapausten automatisoinnille. Signaloinnin jaostossa automatisointia oli jo aiemminkin mietitty ja todettu selkeä tarve toimintotestauksen automatisoinnille. Kokonaisvaltainen testauksen automatisointi kattaa laajasti sekä testitapausten suorituksen että analysoinnin automatisoinnin. Tämän työn tarkoituksena on toteuttaa ainoastaan projektin ensimmäinen vaihe eli analysoinnin automatisoinnin pilotointi valituilla regressiotestitapauksilla IDA2-AA työkalulla. Päätestauksessa tuloksia tarkastellaan vielä toistaiseksi manuaalisesti.

Analysoinnin automatisoinnin hyödyt halutaan mahdollisimman nopeasti käyttöön, joten projektin aikataulu on määritelty melko tiukaksi. Tässä työssä aloitettua automatisointia on tarkoitus jatkaa koko regressiotestijoukon analysoinnin automatisoinnilla. Kyseinen joukko sisältää n. 40 testitapausta.

Analysoinnin automatisointi on uusi työvaihe projektissa, joten sen käyttöönotto vaatii paljon selvitettävää ja käytännön asioiden hoitamista. Projektiin liittyen perustettiin työryhmä, johon osallistui yksi henkilö jokaisesta projektiin osallistuvasta jaoksesta. Työryhmän tarkoituksena on toimia tiedon ja kokemusten jakajana. Työryhmässä tehdään myös tarvittavia päätöksiä liittyen automaatioprojektin analysoinnin osuuteen.

Projekti aloitettiin tammikuussa 2006 pidetyllä koulutuksella. Koulutus koski analysoinnin automatisointiin suunniteltua työkalua. Sen pohjalta jokainen jaos sovelsi automatisointia omiin testitapauksiinsa. Työryhmän kesken, raportoitiin kokemuksista ja työn edistymisestä. Ennen automatisoinnin aloittamista, työryhmässä käytiin kuitenkin käytännön asioita läpi. Ensimmäisenä asiana oli tiedostojen nimeäminen. työryhmässä mietittiin yhteistä nimeämiskäytäntöä, jotta toisten tekemiä skriptejä (komentosarjoja, joilla automatisoidaan tehtäviä ilman, että tarvitaan varsinaista ohjelmointikieltä) voitaisiin käyttää hyväksi. Samoin tiedostojen tallennuspaikka sovittiin yhteisesti. Skriptien uudelleenkäytettävyyttä

mahdollistettiin sopimalla, että skripteistä tehdään mahdollisimman dynaamisia esimerkiksi käyttämällä dynaamisia muuttujia.

5.5 Automatisoinnin kehys

Laaja toimintotestauksen automatisointi sisältää suorituksen-, analysoinnin- sekä raportoinnin automatisoinnin. Signaloinnin osalta testisuoritukset ovat osaksi automatisoitu HIT-ohjelman makroilla. Käytännössä testitapauksia voidaan suorittaa ajamalla makroja, olettaen tietenkin, että konfiguraatio ja muut määrittelyt ovat kunnossa. Testitapauksen suorituksen jälkeen tuloksia verrataan odotettuihin tuloksiin. Tulosten tarkastelua eli analysointia ei tämän projektin alussa ollut automatisoitu ollenkaan signaloinnin osalta. Myös tulosten raportointi on aina tapahtunut täysin manuaalisesti. Aikaisemmin tehdyn kartoituksen perusteella todettiin, että ongelmana nykytilanteessa on erot testaustavoissa testaajien kesken sekä puuttuva yhtenäinen toimintatapa tulosten analysoinnille. Ongelman ratkaisemiseksi toimintatapoja tulee selkeyttää ja yhtenäistää.

Tässä työssä testitulosten analysoinnin automatisointiin käytetään IDA2-AA ohjelmaa. Automatisointityökalun makrot muodostuvat testitoimenpiteistä (test step, sisältävät koodin) ja ohjaustiedostoista (control file) sekä mahdollisista projekteista (project). Toimenpiteet tallentuvat *.ias–tiedostoihin, ohjaustiedostot *.iac–tiedostoihin ja projektit *.iap–tiedostoihin. Makrojen teossa apuna käytetään funktiokirjastoja, jotka tallentuvat *.iaf-tiedostoihin. Analysointien tulokset eli raportit tallentuvat *.rep–tiedostoihin.

Signalointijaoksen osalta päätettiin, että analysointimakrot tullaan tallettamaan verkkolevylle. Makrot tallennetaan kansioihin ominaisuuksittain ja niiden alle testitapauksittain. Samaan hakemistoon tehdään myös funktiokirjasto useasti käytetyille funktioille, näin helpotetaan makrojen tekoa ja muokkausta jatkossa. Kirjastoon tallennetaan funktioita automatisointityön edetessä. Makrojen nimeämiskäytännöksi sovittiin yksinkertaisesti testitapauksen identifiointi-numero,

esimerkiksi 697005001.ias. Kirjasto nimetään niin, että nimen alussa on toimintotestausta kuvaava lyhenne, jaoston tunnus sekä numerointi, FT_Signalling_Library_xx. Lisäksi suunniteltiin jokaisen kirjaston ja sen sisällä olevien funktioiden alkuun tulevat kommenttikentät. Näin ollen muiden on helppo tarkastella ja mahdollisesti muokata tehtyjä tiedostoja. Kuvassa 5.4 kirjaston alussa käytettävät kentät: kirjaston tarkoitus ja kuvaus, mahdollisen ominaisuuden identifiointi numero ja nimi (mikäli kirjasto liittyy suoraan johonkin tiettyyn ominaisuuteen), päiväys, tekijän nimi ja muutoksen tekijän tiedot. Funktioiden kommenttikentässä (kuva 5.5) on muuten samat tiedot, mutta lisäksi käytettyjen parametrien nimet ja tyypit sekä palautettujen arvojen tyypit.

```
////////////////////////////////////////////////////////////////////////////////////////
//                                                                                    //
//   PURPOSE        :     Purpose of this library                                     //
//   DESCRIPTION    :     Specific description of this library                        //
//   AA PLAN        :     Possible feature number and name                            //
//                                                                                    //
//   DATE           :     dd/mm/yyyy                                                  //
//   AUTHOR         :     Author's name and section                                   //
//   CHANGED BY     :     Changer's name and date                                     //
//                                                                                    //
////////////////////////////////////////////////////////////////////////////////////////
```

Kuva 5.4 IDA2-AA kirjaston alkukommentti

```
/*************************************************************** *
*   FUNCTION       :     Function's name                          *
****************************************************************
*   PURPOSE        :     Purpose of this function                 *
*   DESCRIPTION    :     Specific description of this function    *
*   PARAMETERS     :     Name, type                               *
*   RETURN VALUE   :     Type                                     *
****************************************************************
*   DATE           :     dd/mm/yyyy                               *
*   AUTHOR         :     Author's name                            *
*   CHANGED BY     :     Changer's name and date                  *
****************************************************************
```

Kuva 5.5 IDA2-AA funktion alkukommentti

5.6 Regressiotestitapausten automatisointi

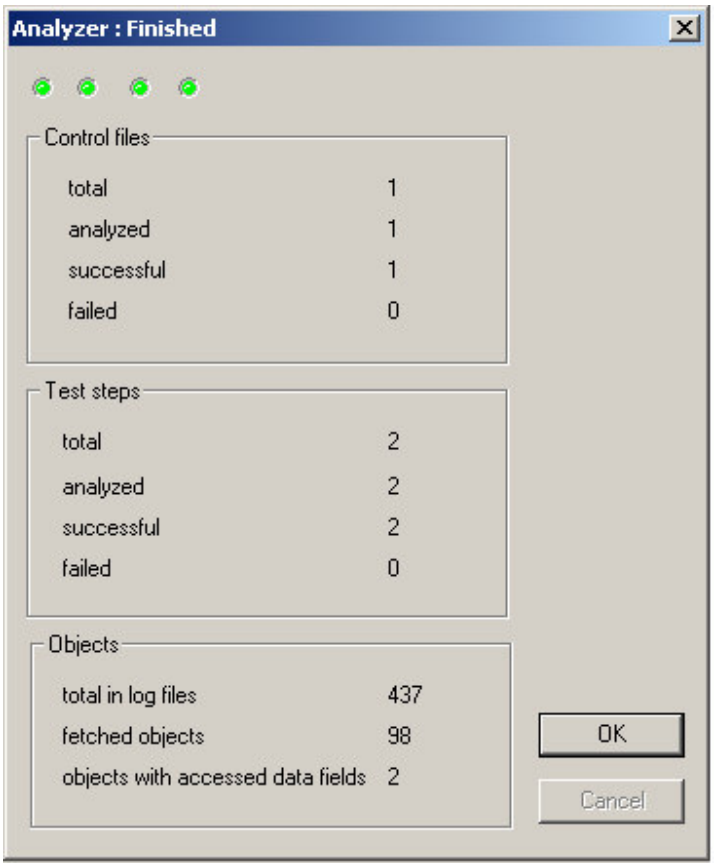
Testauksessa ongelmaksi havaittu toimintamallin puuttuminen ratkaistaan tulosten automatisoinnilla, jolloin tulokset analysoidaan aina samalla tavalla testitapauskohtaisesti. Koska sanomien kulkua voidaan seurata eri rajapinnoista, ennen automatisoinnin aloittamista määriteltiin yhtenäinen linja sanomien tarkastelulle. Linjauksen mukaan tarkastellaan ainoastaan MSC:n sisältä ulospäin lähteviä sanomia. Linjauksen lisäksi testitapausten kuvaukset on oltava mahdollisimman selkeitä analysoinnin automatisoinnin mahdollistamiseksi.

Edellä kuvattuun kehykseen perustuen testaustyöryhmän ohjeistettiin selkeyttämään testitapausten dokumentointia. Dokumentoinnin selkeyttäminen oli osa edellistä uudelleensuunnitteluprojektia, jonka tarkoituksena oli suunnitella uudelleen ohjelmiston ominaisuuksiin liittyvät testitapaukset. Automatisoinnin kannalta on tärkeää, että valitut testitapaukset ovat havainnollisia ja hyvin suunniteltuja. Näin ollen, uudelleensuunnittelun ohella jokainen testaaja valitsi parhaat testitapaukset automatisoitavaan regressiotestijoukkoon. Uudelleensuunnittelun tuloksena saatiin 40 testitapauksen regressiotestijoukko. Regressiotestijoukosta valittiin testitapaukset tämän työn automatisoinnin pilotointia varten. Automatisointiin valittiin luonteeltaan erityyppisiä testitapauksia eri signaloinneista (mm. ISUP, IUP, BICC). Projektin alussa pidetyn koulutuksen sekä automatisointityöryhmässä tehtyjen sopimusten (mm. nimeämiskäytäntö) pohjalta aloitettiin automatisointi. Seuraavissa kappaleissa käydään läpi yhden testitapauksen automatisointimakron tekeminen.

IDA2-AA:han luodaan uusi työtiedosto (workspace), johon määritellään testitulosten monitorointitiedostot ja mahdolliset muut lokitiedostot. IDA2-AA työkalu avaa sanomamonitoroinnin omaksi ikkunaksi, jossa nähdään riveittäin aikajärjestyksessä sanomien kulku eri ohjelmalohkojen välillä (liite 1). Automatisointimakroa varten luodaan uusi testitoimenpide (test step). Monitorointi-ikkunasta raahataan ”drag and drop” -menetelmällä testitapauksen kannalta oleelliset viestit toimenpidesekvenssiin. Toimenpidesekvenssissä

signalointisanomille voidaan esimerkiksi määritellä sekvenssimalli. Sekvenssimallilla määritellään viestien järjestys ja mahdolliset viestien väliset sanomat. Lisäksi jokaisen viestin sisältöä voidaan tarkastella. Viestistä voidaan esimerkiksi tarkastaa, että jollakin tietyllä parametrilla on jokin tietty arvo. Tähän liittyvä skripti kirjoitetaan toimenpidesekvenssiin (liite 2). Kyseiselle esimerkkitapaukselle tehtiin kaksi testitoimenpidettä, joista toinen tarkastelee sanomamonitorointitiedostoa ja toinen protest-testeriltä saatua lokitiedostoa. Nämä testitoimenpiteet kootaan yhteen ohjaustiedostolla (control file), jolla voidaan esimerkiksi määritellä muuttujia ja testitapauksen ehtoja. Testiskriptiä voidaan kirjoittaa myös manuaalisesti, mutta käyttöliittymän kautta ohjelma tekee sen myös automaattisesti. (Liite 3)

Esimerkkitapauksen makro koostuu kahdesta testitoimenpiteestä ja yhdestä ohjaustiedostosta. Analysoinnin automatisointi voidaan suorittaa testitapaukselle ajamalla kyseinen makro luotua työtiedostoa vasten. Analysoinnin jälkeen avautuvasta ikkunasta (kuva 5.6) nähdään heti analysoinnin tulos. Analysointituloksesta nähdään suoritettujen ja analysoitujen ohjaustiedostojen määrä, analysoitujen testitoimenpiteiden määrä sekä onnistuneiden ja epäonnistuneiden tiedostojen määrä. Lisäksi saadaan vielä erillinen raportti analysoinnin tuloksista, jossa on koottuna kaikki tiedot suoritetusta analysoinnista sekä hyväksytty- tai hylätty-tieto (liite 4).



Kuva 5.6 Analysointitulos

Analysointimakrot tehtiin noin kymmenelle testitapaukselle. Tuloksena saatiin pilotoitua toimivat esimerkkitapaukset signaloinnin toimintotestauksen analysoinnin automatisoinnille. Eri signaloinneissa analysointi ei tuottanut ongelmia. Voidaan siis todeta menetelmän soveltuvan signaloinnin tarpeisiin. Jatkosuunnitelmana on analysoida loput 30 regressiotestitapausta ja ottaa automatisoidut testitapaukset käyttöön heti seuraavassa regressiossa, elokuussa 2006. Tämän analysoinnin automatisoinnin pohjalta on mahdollista jatkaa eteenpäin automatisoimalla koko regressiotestauksen kulku. Seuraava vaihe olisi yhdistää ylemmän tason ohjelmalla testisuorituksen- ja analysoinnin automatisointi ja liittää siihen vielä automaattinen raportointi. Tämän toteutuksen jälkeen regressiotestaus voitaisiin periaatteessa suorittaa ns. ”nappia painamalla” ja hyväksytty- tai hylätty-tieto saataisiin tulosteena. Näin laaja automatisointi vaatii kuitenkin paljon suunnittelua ja määrittelyä, jotta siitä saadaan tarpeeksi luotettava.

6 PÄÄTELMÄT

GSM-verkon ja tietoliikennealan viimevuosien huima kehitys sekä tulevaisuuden kehitysnäkymät tuovat kehittämistarpeita sen joka osa-alueelle. Kehittämistarpeista johtuen testauksen osa-alue on yksi tärkeä kehittämiskohde ohjelmistotuotannossa.

Tämän työn tekeminen edellytti perehtymistä GSM-verkon signalointiin ja sen testaukseen. Kirjallisuuden avulla on tutustuttu ohjelmistotestauksen periaatteisiin ja sitä kautta testauksen automatisoinnin tuomiin mahdollisuuksiin. Myös työvälineisiin perehtyminen ja niiden ymmärtäminen on ollut yksi edellytys työn etenemiselle. Esimerkiksi IDA2 työkaluna, jolla testitapausten automatisointi tehtiin, on tullut varsin tutuksi tämän työn aikana.

Työ on osa laajempaa automatisointiprojektia ja sijoittuu projektin alkuun. Työn alussa tehtiin signaloinnin toimintotestauksen kartoitus, jonka tuloksena todettiin, että testaustavoille ja testaustulosten hyödyntämiselle ei ollut yhteistä toimintatapaa. Näin ollen testauskäytäntöjä päätettiin selkeyttää ja yhtenäistää. Yhteistyössä muiden projektin osallisten kanssa tehdyt linjaukset ovat antaneet suunnan ja selkeät rajat tälle työlle. Tarkoituksena tässä työssä oli myös automatisoida valittujen regressiotestitestitapausten analysointi. Jatkosuunnitelmana on tehdä sama koko regressiotestijoukolle. Tulevaisuudessa myös päätestauksen analysointi on mahdollista automatisoida. Tämän työn jatkosuunnitelmana on myös laajemman automatisointijärjestelmän rakentaminen. Periaatteena siinä on, että pystyttäisiin kokoamaan yhteen sekä testauksen automatisointi että tulosten analysoinnin automatisointi. Näin saataisiin käytettävissä olevat resurssit vielä tehokkaammin käyttöön.

Automatisoidun testauksen hyötyjä tarkasteltaessa, olemassa olevien testien suoritus uusissa ohjelmistoversioissa on eräs tärkeimmistä. Hyvin suunnitellut ja tehdyt testitapaukset ovat helposti ja nopeasti siirrettävissä uuteen ohjelmistoversioon. Tästä on hyötyä varsinkin, jos ohjelmisto muuttuu usein.

LÄHTEET

Painetut lähteet

1 SYSTRA System Training, Nokia Networks Oy, Nokian sisäinen materiaali

2 Ron Patton, Software Testing, second edition, Sams Publishing USA, 07/2005, 389s

3 Tapio Hämeen-Anttila, Tietoliikenteen perusteet

4 Glenford J. Myers, The Art of Software Testing, Second Edition, John Wiley & Sons, Inc., Hoboken, New Jersey, 2004. 234s.

5 Jyrki Penttinen, GSM-tekniikka, Järjestelmän toiminta, palvelut ja suunnittelu, 1. painos, Werner Söderström Osakeyhtiö, Porvoo 1999. 349s.

6 Kirsi Willa - Seppo Uusitupa, Tietoliikenneaapinen, Teletekniikkaa ymmärrettävästi, neljäs korjattu painos, Tammer-Paino Oy, Tampere 2001.218s.

7 Ilkka Haikala – Jukka Märijärvi, Ohjelmistotuotanto. Kymmenes, uudistettu painos. Talentum, Helsinki 2004. 440s.

8 Edward Kit, Software Testing in the real world, improving the process. ACM Press, Iso-Britannia 1995. 252 s.

9 William E. Lewis, Software Testing and Continuous Quality Improvement, second edition. CRC Press LLC, USA, 2005. 534 s.

10 Louise Tamres, Introducing Software Testing, Pearson Education Limited, Iso-Bitannia 2002. 281 s.

11 Cem Kaner - David Pels, Bad Software, What to do when software fails. Wiley Computer Publishing, USA 1998. 365 s.

12 Martin Pol - Ruud Teunissen - Erik van Veenendaal, Software Testing, A guide to the Tmap Approach. IQUIP informatica B.V. Iso-Britannia 2002. 564 s.

13 Mark Fewster - Dorothy Graham, Software Test Automation, Effective use of test execution tools. ACM Press, Iso-Britannia 1999. 574 s.

Painamattomat lähteet

14 Nokia Networks Oy, sisäinen materiaali

15 TietoEnator Telecom&Media Oy, sisäinen koulutusmateriaali.

Sähköiset lähteet

16 Peter Rostas, Short Message Services (SMS) liikenteenhallinta, S-38.201 ATM ja Multimedia Seminaari, syksy -96. [www-sivu]. http://www.netlab.tkk.fi/opetus/s38201/k97/rostas.pdf

17 Harri Välimäki, SS7 – Keskusten välinen merkinanto. [www.sivu]. http://www.netlab.tkk.fi/opetus/s38117/k99/Esitelmat/harri_valimaki.pdf

18 Cisco Systems, Inc. ISUP and TCAP [www-sivu] [viitattu 27.5.2006] http://www.cisco.com/univercd/cc/td/doc/product/tel_pswt/vco_prod/ss7_fund/ss7fun05.htm

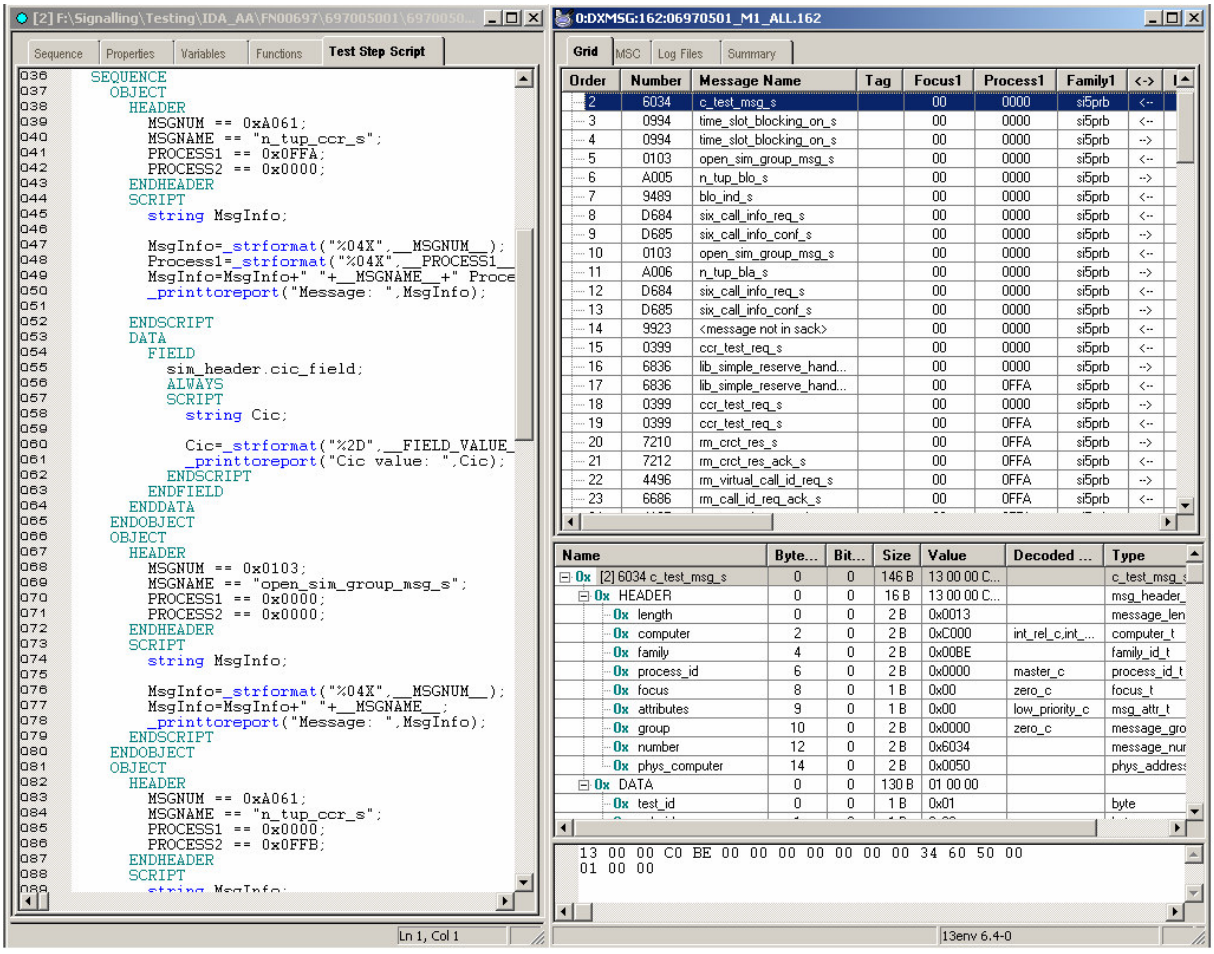
19 Harri Lindberg, Extreme Programming. Tampereen yliopisto, Pro gradu -tutkielma,2003. http://www.cs.uta.fi/research/thesis/masters/Lindberg_Harri.pdf

20 Mika Koskela, Mittaaminen luotettavien ohjelmistotuotteiden kehitysvälineenä, Diplomityö. Teknillinen korkeakoulu, Espoo 2004. [viitattu 31.5.2006] http://www.sal.tkk.fi/English/Publications/pdf-files/TKOS04.pdf

LIITTEET

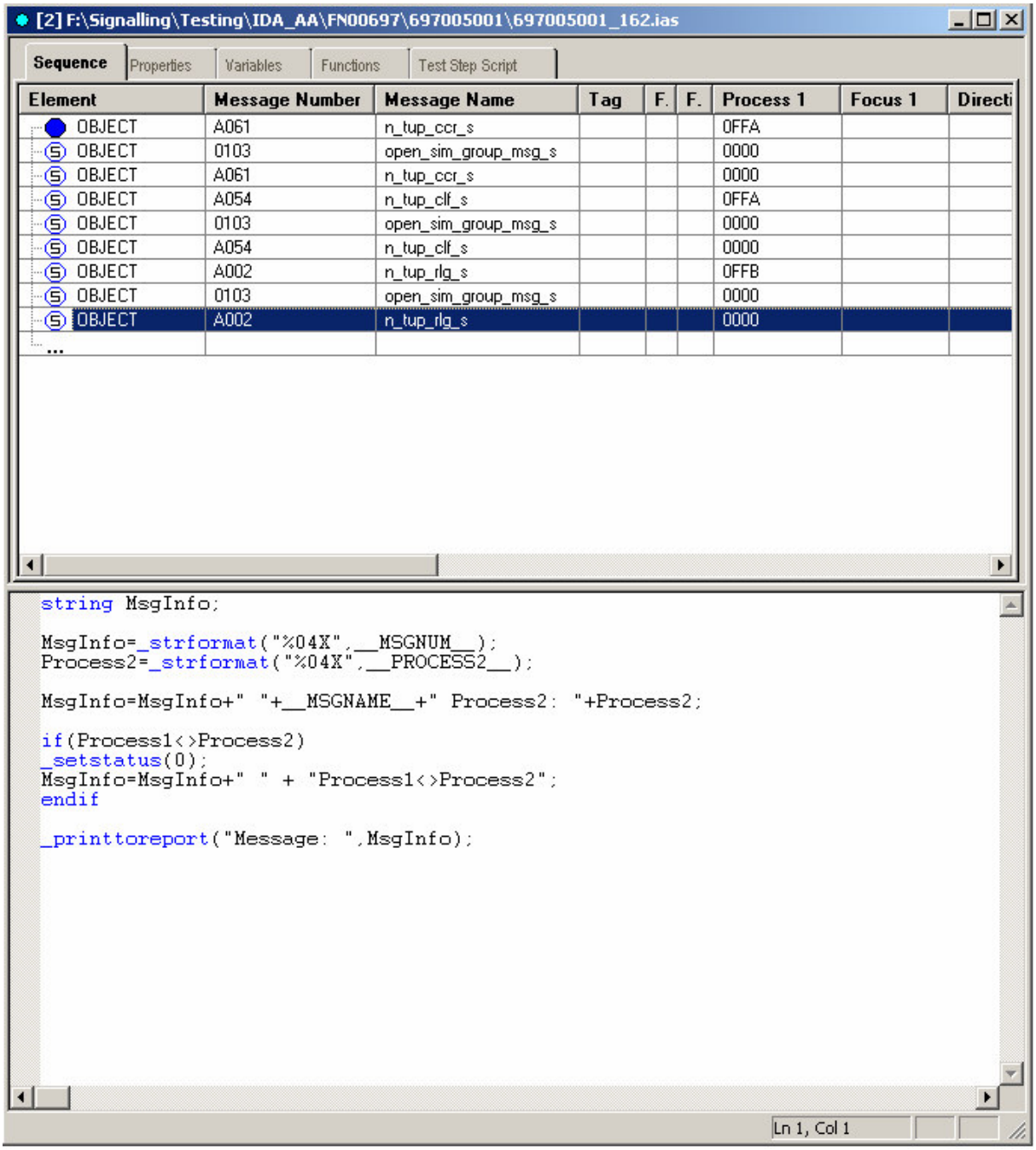
Liite 1. IDA2-AA sanomamonitorointi ja automatisointiskripti

Näkymä IDA2-AA:n sanomamonitorointi-ikkunasta (oikea puoli) sekä osa automatisointiskriptiä (vasen puoli).



Liite 2. IDA2-AA testitoimenpide

Näkymä IDA2-AA toimenpidesekvenssistä ja sen alapuolella olevasta skripti-osasta.

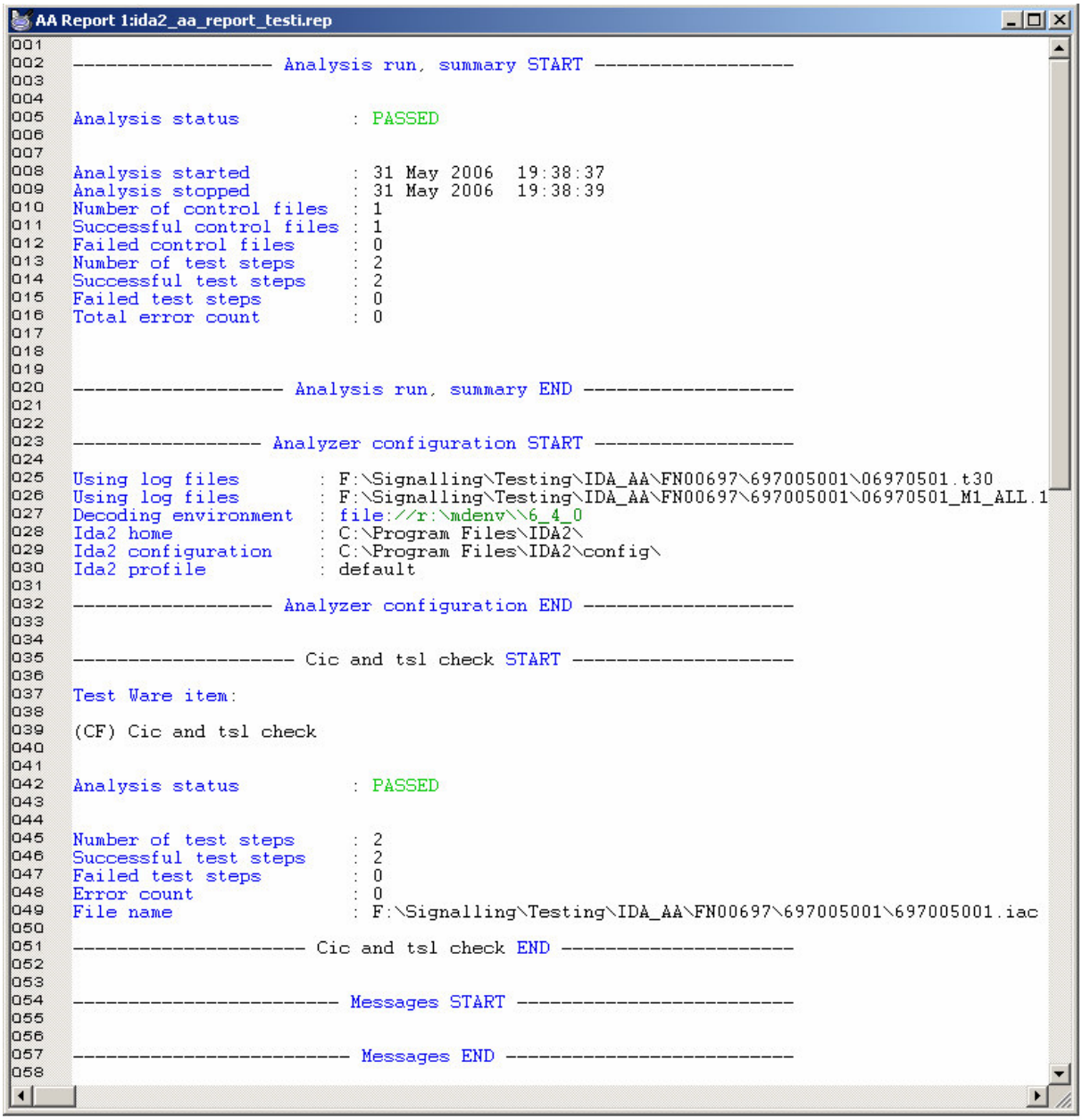


Liite 3. IDA2-AA ohjaustiedoston testiskripti

Esimerkki ohjaustiedoston testiskriptistä esimerkkitestitapaukselle.

```
TESTSTEPCONTROLFILE
  MODULEINFO
    Name = "Cic and tsl check";
    Version = "1.0  10.03.2006";
    Author = "Teija Setola";
    Description =
    "Checks that cic = tsl";
  ENDMODULEINFO

  VARIABLES
    INPUTVARIABLES
    ENDINPUTVARIABLES
    OUTPUTVARIABLES
    ENDOUTPUTVARIABLES
    LOCALVARIABLES
      string Cic = "0";
      string Tsl = "0";
    ENDLOCALVARIABLES
  ENDVARIABLES

  FUNCTIONLIST
  ENDFUNCTIONLIST

  TESTSTEPLIST
     "F:\\Signalling\\Testing\\IDA_AA\\FN00697\\697005001\\697005001_162.ias";
     "F:\\Signalling\\Testing\\IDA_AA\\FN00697\\697005001\\697005001_t30.ias";
  ENDTESTSTEPLIST

  SEQUENCE
    TESTSTEP
      Name = "Msg sequence check";
    ENDTESTSTEP
    TESTSTEP
      Name = "Tsl check";
    ENDTESTSTEP
    ACTION
      Title = "Check CicValue";
      SCRIPT
        if(Cic<>Tsl)
        _setstatus(0); //marks the cf failed
        _printtoreport("Cic <> Tsl","");
        endif
      ENDSCRIPT
    ENDACTION
  ENDSEQUENCE
ENDTESTSTEPCONTROLFILE
```

Liite 4. IDA2-AA raportti



AA Report 1:ida2_aa_report_testi.rep

```
001
002  ------------------ Analysis run, summary START ------------------
003
004
005  Analysis status          : PASSED
006
007
008  Analysis started         : 31 May 2006  19:38:37
009  Analysis stopped         : 31 May 2006  19:38:39
010  Number of control files  : 1
011  Successful control files : 1
012  Failed control files     : 0
013  Number of test steps     : 2
014  Successful test steps    : 2
015  Failed test steps        : 0
016  Total error count        : 0
017
018
019
020  ------------------- Analysis run, summary END -------------------
021
022
023  ------------------ Analyzer configuration START ------------------
024
025  Using log files        : F:\Signalling\Testing\IDA_AA\FN00697\697005001\06970501.t30
026  Using log files        : F:\Signalling\Testing\IDA_AA\FN00697\697005001\06970501_M1_ALL.1
027  Decoding environment   : file://r:\mdenv\\6_4_0
028  Ida2 home              : C:\Program Files\IDA2\
029  Ida2 configuration     : C:\Program Files\IDA2\config\
030  Ida2 profile           : default
031
032  ------------------- Analyzer configuration END -------------------
033
034
035  --------------------- Cic and tsl check START ---------------------
036
037  Test Ware item:
038
039  (CF) Cic and tsl check
040
041
042  Analysis status          : PASSED
043
044
045  Number of test steps     : 2
046  Successful test steps    : 2
047  Failed test steps        : 0
048  Error count              : 0
049  File name                : F:\Signalling\Testing\IDA_AA\FN00697\697005001\697005001.iac
050
051  ---------------------- Cic and tsl check END ----------------------
052
053
054  ------------------------ Messages START ------------------------
055
056
057  ------------------------- Messages END -------------------------
058
```